ÇANKAYA UNIVERSITY FACULTY OF ENGINEERING

CENG 408

Innovative System Design and Development Ⅱ Project Report

Spaceborn

Aleyna Saykılı 201911054

Aslı Ceren Hızır 202128035

Dilara Bayındır 202011079

Ata Doruk Çakmak 202111030

Oğuz Akay 202128011

Advisors: Murat SARAN- Talha KARADENİZ

Table of Contents

1. Introduction vi
2. Project Plan vi
3. Literature Review vii
3.1 öz vii
3.2 Abstract viii
4. Used Technologies ix
4.1 Risks x
4.2 Existing/Similar Solutions x
5. Conclusion xii
Software Requirements Specification (SRS) xii
1. Introduction xii
1.1 Purpose xii
1.2 Scope xiii
1.2.1 Software Product Identification xiii
1.2.2 Software Product Capabilities xiii
1.2.3 Application and Benefits xiii
1.2.4 Consistency with Higher-Level Specifications xiv
1.3 Glossary xiv
1.4 Overview xv
1.5 Version History xv
2. Overall Description xv
2.1 Product Perspective xv
2.1.1 System Interfaces xv
2.1.2 User Interfaces xvi
2.1.3 Hardware Interfaces xvi
2.1.4 Software Interfaces xvii
2.1.5 Communications Interfaces xvii
2.1.6 Memory Constraints xvii
2.1.7 Operations xvii
2.1.8 Site Adaptation Requirements xviii
2.2 Product Functions Procedural Map Generation: xviii
2.3 User Characteristics xix
2.4 Constraints The development of Spaceborn is subject to the following constraints: xx
2.5 Assumptions and Dependencies xx
2.6 Apportioning of Requirements xxii
3. Specific Requirements xxii

3.1 External Interface Requirements xxii
3.2 Functional Requirements xxiii
3.2.1 Save Game Functionality xxiii
3.2.3 Resource Management xxiv
3.2.4 Main Menu Use Case xxiv
3.2.5 Gameplay Use Case xxv
3.2.5.1 Brief Description: xxv
3.2.5.2 Initial Step-by-Step Description: xxvi
3.2.6 In-Game Settings Menu Use Case xxviii
3.2.6.1 Brief Description xxviii
3.2.6.2 Initial Step-by-Step Description xxviii
3.3 Error Handling xxix
3.4 Performance Requirements xxx
3.5 Logical Database Requirements xxx
3.6 Design constraints xxxi
3.6.1 Standards compliance xxxii
3.7 Software System Attributes xxxii
3.7.1 Reliability xxxii
3.7.2 Availability xxxii
3.7.3 Security xxxiii
3.7.4 Maintainability xxxiii
3.7.5 Portability xxxiii
Software Design Description (SDD) xxxiv
1.Introduction xxxiv
1.1 Purpose: xxxiv
1.2 Definitions: xxxiv
1.3 Overview xxxiv
1.4 Glossary xxxv
2. System Overview: xxxv
2.1 System Objectives xxxv
2.2 System Architecture xxxv
3. Data Design xxxv
3.1 Database Design xxxv
3.2 Data Security xxxvi
3.3 Data Backup and Recovery xxxvi
3.4 Activity Diagram xxxviii
3.5 Sequence Diagram xxxviii
3.5.1 Login and Initialize Game xxxviii

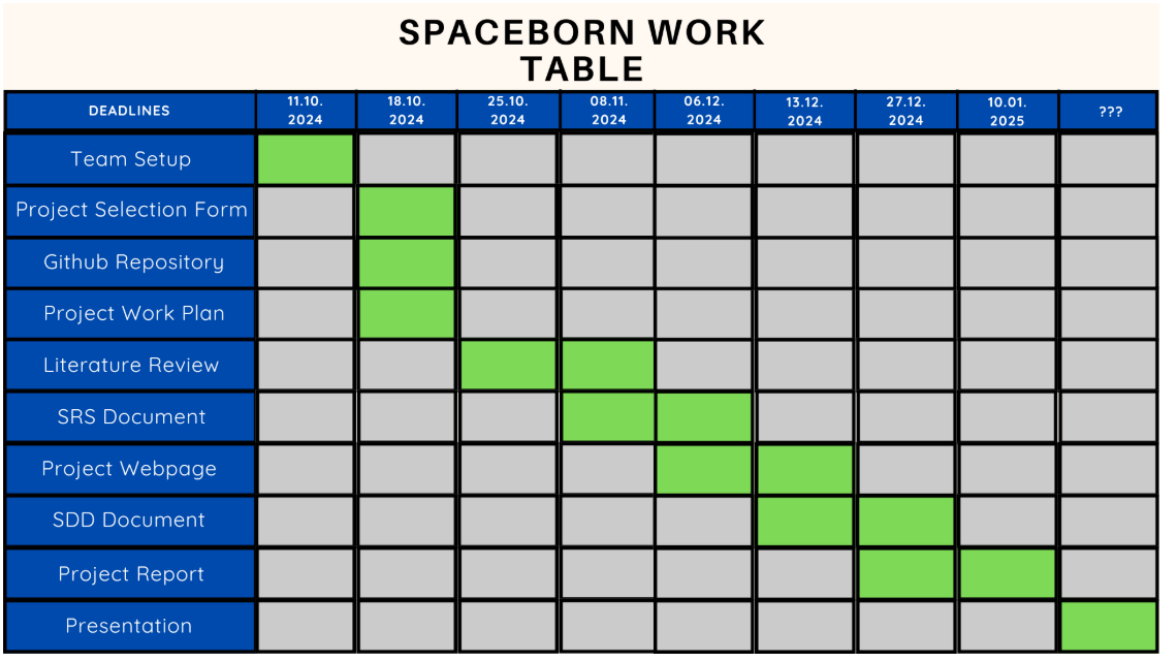
3.5.2 Oxygen Management Puzzle xxxix
3.5.3 Oxygen Management and Colony Status xxxix
3.5.4 Food Management Wheel xl
3.5.5 Temperature Management xl
3.5.6 Embryo Health Check xli
3.5.7 Start Game Sequence xli
3.5.8 Mutant Attack Scenario xlii
4. Interface Design xlii
References xliv
INTRODUCTION TEST PLAN 1
Version Control 1
Overview 1
Scope 1
Terminology 1
FEATURES TO BE TESTED 2
FEATURES NOT TO BE TESTED 2
ITEM PASS/FAIL CRITERIA 2
Exit Criteria 2
TEST RESULT 3
1. Test Objectives 3
2. Test Environment 3
3. Tested Features 3
4. Bug Reports & Fixes 5
5. Conclusion 5
REFERENCES 6
https://github.com/CankayaUniversity/ceng-407-408-2024-2025-Spaceborn/wiki 6
TEST DESIGN SPECIFICATIONS 6
6.1 Test Cases 6
LG.AD.01 7
LG.AD.02 7
TEST RESULT 14
1. Test Objectives 14
2. Test Environment 14
3. Tested Features 15
4. Bug Reports & Fixes 16
5. Conclusion 17
USER MANUEL 17
1. Introduction 17

2. System Requirements 17
3. Installation 18
4. User Interface 18
5. Basic Functions 18
6. Running the Simulation 18
7. Performance Evaluation.....19
8. Troubleshooting.....19
9. Frequently Asked Questions 19

1. Introduction

This document provides a comprehensive overview of the software design of the Spaceborn project. It is a technical guide aimed at clarifying the project, system architecture, technical components and data. Spaceborn is a survival and colony management game set in space, providing management of game resources, building systems and protecting colonies from external threats. The sections in this document cover advanced features and usage scenarios, explaining the general storage and storage of the system step by step. The literature review provides background information supporting the goals of Spaceborn by introducing the basic collection and comprehensive features. The Software Requirements Specification (SRS) section clarifies the goals, parameters and general features of the game. The Software Design Document (SDD) details the architecture, features and design elements that are deployed from the user perspective. This document provides a comprehensive guide for presenting and explaining the Spaceborn project step by step.

2. Project Plan



SPACEBORN WORK TABLE

DEADLINES	11.10. 2024	18.10. 2024	25.10. 2024	08.11. 2024	06.12. 2024	13.12. 2024	27.12. 2024	10.01. 2025	???
Team Setup									
Project Selection Form									
Github Repository									
Project Work Plan									
Literature Review									
SRS Document									
Project Webpage									
SDD Document									
Project Report									
Presentation									

3. Literature Review

3.1 öz

Spaceborn projesi, uzay temalı bir koloni inşa etme ve hayatta kalma oyunu olarak tasarlanmıştır. Bu oyun strateji ve puzzle temalarını bir araya getirerek oyuncuları içine çekmeyi hedefler. Oyuncular prosedürel olarak oluşturulmuş bir uzay gemisinde kolonilerin hayati sistemlerini kontrol ederken bir yandan da düşman mutantlara karşı

savunma yapar. Oyuncu kolonilerin hayati sistemini onların oksijen seviyesini, gıda ihtiyaçlarını ve enerji yönetimlerini dengeleyerek sağlar. Ayrıca prosedürel harita sayesinde de oyuncu her oynayışta farklı deneyimler elde edebilir. Bu oyunu geliştirebilmek için Unreal Engine oyun motoru kullanılacaktır. Kodlama için C++ dilinin performans avantajlarından ve Unreal Engine'nin görsel programlama aracı olan blueprint ile hızlı oyun mekaniği oluşturma avantajından yararlanılacaktır. Ek olarak tasarım ve modellemeler için Blender ve Photoshop uygulamaları kullanılacaktır. Projeyi geliştirirken teknik riskler, kaynak yetersizliği ve en önemlisi market riski göz önünde bulundurulacaktır. Bunun için piyasadaki benzer oyunlar araştırılıp onlardan farklı olarak sunabileceğimiz oyun mekaniği, oyun tasarımı ve oyunun hikayesi üzerine durulacaktır.

3.2 Abstract

The Spaceborn project is designed as a space-themed colony building and survival game. This game aims to captivate players by combining strategy and puzzle elements. Players manage the vital systems of colonies aboard a procedurally generated spaceship while defending against hostile mutants. The player ensures the colony's vital systems by balancing oxygen levels, food needs, and energy management. Additionally, thanks to the procedural map, players can experience different challenges with each playthrough. This game will be developed using the Unreal Engine game engine. The performance advantages of C++ will be utilized for coding, while the visual programming tool, Blueprint, will be leveraged to quickly create game mechanics. Additionally, Blender and Photoshop will be used for design and modeling. While developing the project, technical risks, resource shortages, and, most importantly, market risks will be taken into account. To address this, similar games in the market will be researched, and the focus will be on offering unique game mechanics, game design, and storyline elements that differentiate the game from others.

Introduction

The gaming industry has become a rapidly growing sector in recent years. With the advancement of technology, games have become more visually impressive and gameplay has become smoother. Divided into branches such as mobile, PC, and console games, it appeals to a wide range of age groups. Every day, new technologies are being developed in this field, allowing it to continue evolving. Strategy games hold an important place among video games. They provide players with opportunities for planning, making tactical decisions, and critical thinking. Players are required to think not only in the short term but also in the long term. Especially with the advancement of technology, artificial intelligence and procedural generation have made the game more dynamic, offering players a different experience with each playthrough.

The Spaceborn project is a game that invites players to establish a colony in space. In this game, the player must meet the essential needs for the colony's survival. Additionally,

they must combat mutants that have infested the spaceship and face various challenges. These challenges include system malfunctions, resource shortages, and health crises within the colony. The game features procedurally generated maps and events, encouraging players to explore new sections of the spaceship and nearby space objects to gather rare resources. This ensures that the game offers a strategy-themed experience and provides a unique playthrough each time. The goal of the game is to create an enjoyable experience that offers strategy and immersive gameplay for fans of sci-fi and survival genres. The core elements of strategy games are as follows: 1.Resource Management: It requires the player to plan how to use limited resources most efficiently. In our game, resource management is done by adjusting the colony's oxygen levels, food needs, and energy levels. By managing these resources, the player ensures the colony's survival. 2.Planning and Timing: Players must make long-term plans and implement them at the right time. In our game, the player must plan how to meet the colony's food needs and ensure that the necessary actions for survival are carried out in time. While doing this, they must also remember the hostile mutants. 3.Defense and Attack Mechanics: Players must develop defense strategies and attack enemies to slow them down or neutralize them. In our game, there will be three different types of enemies. The player must develop defense and attack strategies based on these enemy types.

4. Used Technologies

Unreal Engine, developed by Epic Games, is a powerful game engine. It will be used as the main game engine for the development of the Spaceborn project. Unreal Engine has attracted the attention of developers due to its high-quality graphics. With C++ integration and Unreal Engine's visual programming tool, Blueprint, game mechanics, animations, AI systems, and UI components are being developed. C++ provides performance advantages for developers, while Blueprint offers fast prototyping and debugging processes. For the design aspects of the game, we will utilize Blender and Photoshop. Blender is a tool used for 3D modeling, animation, and visual effects development. It will be used in the Spaceborn project for environmental design, the design of the game character and enemy characters, as well as for creating their in-game animations. Photoshop is used for 2D assets. It will be used for menu designs, icon designs, and environmental designs within the game. With these tools, we will create a visually impressive game with dynamic animations. Git is a version control system commonly used in game development projects. By using this system, multiple developers can work simultaneously, have the ability to back up and revert code. In our project, Git version control will be used, allowing the team to work together while tracking the development and changes of the project. Additionally, it will offer the ability to easily revert back if any issues arise. Procedural generation algorithms will be used to randomly

generate maps and events. AI systems will also be implemented to code NPC behaviors and mechanics, providing a dynamic experience for the player.

4.1 Risks

1.Technical Risks: The biggest technical challenges in the project are the complexity of procedural generation and AI behaviors. Procedural generation of the map may cause errors in the systems to be developed. In addition, additional errors may occur depending on the unexpected behavior of the player. Apart from this, performance problems may occur. When using high and quality visuals, we need to consider system requirements in terms of performance. In order to reduce these risks, we will start the development process with the procedural map first and try to solve possible errors in advance. In addition, we aim to detect and solve possible problems that may occur early by testing the game at every stage.

2.Resource Risks: The most important resource risk in our game is that team members have insufficient knowledge on some subjects. For example, in our project, only one person is familiar with the animation and modeling part, but for a game, there are too many components to be modeled and animated, so we will try to solve this problem by getting support from ready-made assets. In addition, artificial intelligence and procedural generation algorithms are also difficult topics for developers. Team members will receive cross-training for these and these gaps will be closed. Another resource risk is that not enough games are tested because the more a game is tested, the more we observe possible errors in advance. We plan to solve this problem by holding a test event in our school and having school students play it.

3.Marketing Risks:

The most important market risk in a game project is the intensity of competition. There are many survival and strategy games on the market, and some games are very well-established and good games. Your game needs to differentiate itself from these games in some way. We aim to offer players a new style of play by creating different game mechanics than the games on the market. We also plan to differentiate the game by offering players a different story from other strategy games with the story of the game. Determining the wrong target audience is also very important in the marketing part of the game. We aim to determine this after doing market research and present it to the right audience. Another marketing risk is receiving negative criticism and reviews. We believe that we can solve this by testing before the game is released and by getting feedback from people.

4.2 Existing/Similar Solutions

1.Subnautica

Solution Description: "Subnautica" is a sci-fi survival and exploration game where players find themselves stranded on an oceanic alien planet. The game focuses on survival elements such as managing food, water, and oxygen while exploring the deep sea, gathering resources, crafting equipment, and uncovering the planet's mysteries.

Shortcomings: "Subnautica" is based on an open-world ocean planet environment, lacking the confined space and colony protection dynamics of a spaceship. In contrast, "Spaceborn" offers a colony-building experience in a limited space, emphasizing resource management within the constraints of a spaceship, as well as defense against mutant threats.

2.Dead Space

Solution Description: "Dead Space" is a sci-fi horror game where players navigate a spaceship overrun by hostile alien-like creatures. It emphasizes a story-driven survival horror experience, with players managing limited resources and battling mutants to survive in a dark, atmospheric environment.

Shortcomings: While "Dead Space" delivers a compelling horror experience, it lacks colony-building, resource management, and open-ended exploration elements. "Spaceborn" differentiates itself by integrating survival, resource management, and a colony-defense strategy into a narrative-driven gameplay, combining elements of survival horror with sci-fi colony management.

3.Frostpunk

Solution Description: "Frostpunk" is a survival and colony simulation game where players manage resources and citizens' needs to survive in a post-apocalyptic frozen wasteland. The game requires strategic decision-making to keep a growing city alive in the face of environmental threats and resource scarcity.

Shortcomings: "Frostpunk" is focused on city-building in a harsh environment on Earth and lacks the unique atmosphere of a space-based setting. "Spaceborn" takes the colony management concept into space, presenting unique challenges within a spaceship environment and combining it with exploration of space objects and defensive mechanics against mutant threats.

4.Alien: Isolation

Solution Description: "Alien: Isolation" is a sci-fi survival horror game where players must avoid a terrifying alien while managing resources and navigating a hostile spaceship. It combines stealth and horror elements in a highly atmospheric environment.

Shortcomings: "Alien: Isolation" is heavily focused on horror and stealth but lacks colony management, resource-building, and defense setup mechanics. In "Spaceborn," players not only defend against threats but also manage resources, set up defenses, and construct systems essential for the survival of a colony, blending survival horror with strategic elements.

5.RimWorld

Solution Description: "RimWorld" is a sci-fi colony simulator where players manage colonists, resources, and defenses while responding to randomly generated events on an alien planet. It focuses on colony management, resource allocation, and responding to environmental threats.

Shortcomings: Although "RimWorld" provides a colony management experience, it is planet-based rather than confined to a spaceship environment. "Spaceborn" offers the added challenge of limited space within a ship, as well as the threat of mutated entities, creating a more intense and strategic survival experience with space exploration opportunities.

5. Conclusion

The "Spaceborn" project offers a unique blend of survival, strategy, and combat mechanics within a confined space-based setting, providing an alternative to existing games by combining resource management, defense, and exploration with a story-driven approach. During the development process, game dynamics and visual elements will be designed effectively with a powerful game engine such as Unreal Engine, performance-oriented languages such as C++, and design tools such as Blender and Photoshop. The "Spaceborn" project, taking into account potential risks and market research, will provide an exciting experience for fans of sci-fi and strategy games.

Software Requirements Specification (SRS)

1. Introduction

1.1 Purpose

The purpose of this document is to provide a detailed overview of the "Spaceborn" project, including its objectives, requirements, technological specifications, risks, and an analysis of existing solutions. It is intended to guide the project development team and stakeholders in understanding and aligning on the goals and direction of the project.

1.2 Scope

1.2.1 Software Product Identification

The software product to be developed is called "Spaceborn". It is a single-player survival and puzzle-based game designed to be played on PCs. The game integrates various systems, including resource management mini-games, AI-driven alien threats, and procedural map generation, to create a unique survival experience in a confined spaceship setting.

1.2.2 Software Product Capabilities

What the software will do:

• Resource Management: Players will solve mini-games (e.g., puzzles for oxygen restoration, power rerouting) to maintain vital spaceship systems and ensure the survival of human embryos onboard.

• Alien Encounters: The game features multiple types of alien threats, including an intelligent machine-learning-driven enemy that adapts and attempts to track the player.

• Procedural Map Generation: Each gameplay session will have a uniquely generated spaceship layout to enhance replayability.

• Solo Experience: The game is designed for single-player use, optimized for smooth performance on mid-range systems.

• Immersive Storyline: Players uncover the backstory of the mutants, the spaceship's mission, and their ultimate objective while trying to survive until help arrives.

What the software will not do:

• The game will not include multiplayer or online functionality.

• It will not feature external modifications or integrations beyond the core gameplay mechanics.

1.2.3 Application and Benefits

Application of the Software:

"Spaceborn" immerses players in a high-stakes survival experience where they must rely on critical thinking, problem-solving, and strategic decision-making to manage resources and avoid alien threats.

Relevant Benefits, Objectives, and Goals:

• Replayability: Procedural map generation and dynamic alien AI behaviors create a unique experience with every playthrough.

• Engaging Gameplay: Mini-games add depth to resource management, making routine survival tasks interactive and challenging.

• Cognitive Challenge: The machine-learning-powered enemy adds unpredictability, testing the player's adaptability and decision-making under pressure.

• Optimization: Designed to deliver a smooth, visually stunning experience even on mid-range systems, increasing accessibility.

• Story-Driven Engagement: A compelling narrative keeps players motivated and emotionally invested in the survival of the characters and their mission.

1.2.4 Consistency with Higher-Level Specifications

• The description aligns with the overarching system requirements, including the integration of mini-games for resource management, the use of Unreal Engine (C++) for development, and Python for machine learning implementation. These ensure the game meets its objectives of delivering an optimized, engaging, and replayable survival experience.

• Similar products such as "Frostpunk" and "Subnautica" serve as inspirations, demonstrating how resource management and survival can be intricately tied to player engagement. Additionally, games like "Alien: Isolation" showcase the tension and unpredictability of being hunted by adaptive AI, a feature reflected in the machine learning-driven alien encounters in "Spaceborn." Lastly, the procedural generation mechanics align with approaches used in games like "No Man's Sky", ensuring each session offers unique and fresh gameplay experiences.

1.3 Glossary

A table of definitions, terms, and abbreviations used in this document

Term	Description
UI	User Interface
AI	Artificial Intelligence
FPS	First Person Shooter
HUD	Heads-Up Display

1.4 Overview

Spaceborn combines strategy, defense, and story-driven gameplay to provide players with a space survival experience. Unlike existing survival games, the game takes place within a confined ship environment and encourages exploration of new procedurally generated sections with space objects.

1.5 Version History

Provide a table of table of changes to this document

Version No	Description of change	Date
1.0	Initial Release	15.10.2024

2. Overall Description

2.1 Product Perspective

Spaceborn is an independent, self-contained software product designed as a single-player survival game. The product is not a component of a larger system and operates as a standalone application. However, it relies on integration between Unreal Engine, C++, and Python for core functionalities such as game mechanics, procedural map generation, and machine-learning-driven alien behaviors.

The game operates under various constraints and interfaces to provide an optimized and seamless gaming experience. A block diagram is included to illustrate the major components of the system, their interconnections, and interfaces.

2.1.1 System Interfaces

Spaceborn will be developed using either Unity or Unreal Engine and will be compatible with the following systems:

• Operating Systems: Windows and macOS, with future plans for Linux compatibility.

• Hardware Requirements: Mid-range gaming PCs with sufficient resources to support real-time rendering and procedural generation.

• Game Engine: Unity: Known for its flexibility and user-friendly environment, Unity supports efficient procedural generation and 2D/3D asset integration. Unreal Engine: Offers high-quality rendering capabilities and advanced physics simulation for a visually immersive experience. Both engines provide the tools required to implement Spaceborn's core mechanics, including procedural map generation, AI systems, and user interfaces. The final decision will depend on the project team's expertise and the specific features prioritized during development.

2.1.2 User Interfaces

The user interface of Spaceborn will be developed using either Unity or Unreal Engine, depending on the final selection for the game engine. The design focuses on user-friendliness and simplicity to ensure an intuitive experience for players. The main menu includes options such as “New Game,” “Load Game,” “Settings,” and “Exit Game.” The Settings screen allows players to adjust parameters like “Music Volume,” “Resolution,” “Display Mode,” and “Control Sensitivity.” The in-game menu provides options for adjusting “Music Volume,” “Resolution,” and “Display Mode” while also featuring buttons for “Save,” “Cancel,” and “Quit Game.”

The gameplay screen is designed for clarity:

• The top of the screen displays critical resources such as oxygen, energy, and food levels.

• The bottom left features a timeline and a minimap for easy navigation and time management.

• Additional functionalities, such as resource allocation, construction, and defense options, are accessible through dedicated buttons located at the bottom of the screen.

• This interface ensures seamless navigation and interaction within the game, enhancing the player's immersion in the Spaceborn environment.

2.1.3 Hardware Interfaces

Spaceborn will utilize standard PC controls and hardware interfaces to provide an optimal gaming experience. The game will require the following:

• PC Devices: Spaceborn will run on devices with Windows or macOS. Future compatibility with Linux-based systems may also be considered.

Supported Devices: Mid-range PCs with the following configuration: Minimum: 8 GB RAM, quad-core processor, and GTX 1050 or equivalent GPU. Recommended: 16 GB RAM, RTX 3060 or higher GPU.

• Input Devices: The keyboard will be used for essential gameplay functions such as movement, shortcuts, and resource management. The mouse will enable navigation,

object selection, and interaction with in-game menus and systems. Future updates may include support for gaming controllers for players who prefer this input method.

• Ports/Protocols: Game saves are stored locally; no external devices or connections are necessary.

2.1.4 Software Interfaces

• Operating System Compatibility:

Spaceborn will be compatible with major desktop operating systems, including Windows and macOS. The game engine, Unity or Unreal Engine, will provide a stable framework for UI creation, rendering, and core game mechanics.

• Procedural Generation: The chosen engine (Unity or Unreal Engine) will manage the procedural generation of maps and events to ensure unique gameplay experiences. Algorithms for randomization and resource placement will be seamlessly integrated.

• Physics Computations: Both Unity and Unreal Engine provide robust physics systems. These will be used to handle object collisions, environmental effects, and crew movement within the spaceship environment.

• Character and Object Animations: The animation tools in Unity or Unreal Engine will ensure smooth transitions and realistic behaviors for characters, objects, and enemies, creating an engaging and visually appealing game.

2.1.5 Communications Interfaces

Spaceborn operates offline and does not require network communications. However, communication protocols between the game engine and Python are facilitated locally through API calls or shared libraries.

2.1.6 Memory Constraints

Primary Memory: Requires a minimum of 8 GB of RAM for smooth gameplay.

Secondary Memory: The game requires approximately 10 GB of storage space for installation and save files.

2.1.7 Operations

Normal Operations:

• Players interactively solve puzzles, manage resources, and evade enemies.

Backup and Recovery:

• Autosave system triggers after completing major objectives, such as solving puzzles or escaping alien encounters.

• Manual save option available during safe zones.

2.1.8 Site Adaptation Requirements

Data Initialization: The procedural map generation requires predefined templates and initialization sequences specific to gameplay themes.

Adaptation Features: The game engine includes scalability settings, allowing it to adjust graphical fidelity based on the system's hardware capabilities.

2.2 Product Functions Procedural Map Generation:

• The game dynamically generates a unique spaceship layout for every playthrough, ensuring replayability and unpredictability.

Resource Management Mini-Games: Players interact with engaging mini-games to manage critical systems such as:

• Oxygen Supply: Maintaining breathable air for survival.

• Power Distribution: Allocating limited energy to different systems.

• Embryo Preservation: Keeping embryos stable and alive under resource constraints.

Machine Learning-Driven Alien AI:

• One of the alien enemies uses a machine learning algorithm to adapt its behavior based on player actions, creating an intelligent and evolving challenge.

Enemy Variants and Combat:

• Different types of alien creatures with distinct abilities attack the player, requiring strategic planning and fast reflexes to survive.

• Combat includes defensive tools, stealth mechanics, and environmental traps.

Survival Mechanics:

• Players must evade enemies while managing health and stress levels, creating a tense and immersive experience.

Progression and Story Unfolding:

• Completing resource tasks and evading threats gradually reveals the mysterious origins of the mutants and the true mission of the spaceship.

Save and Load System:

• Autosave functionality ensures progress is retained after critical events. Manual saves are available in safe zones.

Optimized Performance:

• The game ensures a smooth experience even on mid-range hardware by offering scalable graphics and efficient memory usage.

Logical Organization of Functions The game functions are interdependent:

• Resource Management directly impacts survival mechanics, as depleted resources increase vulnerability to alien threats.

• Procedural Map Generation creates a unique environment that dictates the placement of resources and enemy behavior.

• Machine Learning AI adjusts based on player strategies, making each playthrough a unique challenge.

2.3 User Characteristics

The intended users of Spaceborn include a diverse audience of gamers with varying levels of experience and technical expertise. The general characteristics of the target user base are as follows:

Educational Level:

• No specific educational qualifications are required, but users should possess basic literacy and comprehension skills to follow tutorials and in-game instructions.

Gaming Experience:

• Beginner to Intermediate Gamers: The game includes intuitive tutorials and guided objectives to accommodate users new to survival or resource management games.

• Experienced Gamers: Procedural maps and machine-learning-driven AI provide unique challenges for seasoned players seeking replayability and strategic depth.

Demographics:

• Age Group: 16 and above, as the game includes intense survival mechanics and suspenseful encounters that may not be suitable for younger players.

• Interest Areas: Fans of survival, science fiction, horror, and strategic resource management games are the primary audience.

Impact on Design and Requirements: The user characteristics influence several aspects of the game's design:

• Ease of Use: Simple and intuitive user interfaces, supported by an optional tutorial mode.

• Scalability: Difficulty levels allow beginners to enjoy the game while providing a challenge to experienced gamers.

• Accessibility: Optimized for mid-range systems to ensure a wide audience can access the game.

• Replayability: Procedural content and evolving AI provide engaging experiences for players seeking variety and depth.

2.4 Constraints The development of Spaceborn is subject to the following constraints:

Technological Constraints

• The game must be developed using either Unity or Unreal Engine, limiting the available development tools and requiring the team to work within the capabilities and limitations of the chosen engine. Procedural generation algorithms and AI systems must be implemented efficiently to ensure real-time performance on mid-range gaming systems.

Resource Constraints

• The team has a limited number of members with specialized skills, such as 3D modeling, AI programming, and game design. Cross-training is required to ensure all critical tasks are addressed. Available hardware and software licenses may limit the tools and resources that can be utilized.

Temporal Constraints

• The project timeline is constrained to a fixed academic schedule, with specific milestones to be completed within weekly sprints and a final deadline for project submission.

Budget Constraints

• Financial resources are limited to available institutional or personal funding, restricting the ability to acquire premium software tools, high-end hardware, or external services.

Design Constraints

• The game must support procedurally generated maps and events while maintaining balanced gameplay and performance. It must also provide an engaging user interface and intuitive controls suitable for a broad audience.

Market Constraints

• The game faces competition from established titles with similar mechanics, such as RimWorld and Subnautica, necessitating a focus on unique features and a compelling gameplay experience. By understanding and addressing these constraints, the Spaceborn team aims to deliver a high-quality product within the available resources and timeline.

2.5 Assumptions and Dependencies

The following assumptions and dependencies are critical for the successful development of Spaceborn and may affect the requirements outlined in this SRS:

• Game Engine Availability and Selection: It is assumed that either Unreal Engine or Unity will be available for development throughout the project lifecycle. If, for any reason, the chosen engine is not available or fails to meet the requirements for gameplay, performance, or machine learning integration, the project scope may need to be reassessed and the development timeline adjusted.

• Machine Learning Libraries and Python Integration: The game will rely on Python for machine learning integration (e.g., AI behavior and procedural generation). It is assumed that necessary libraries (such as TensorFlow, Keras, or PyTorch) will be compatible with the chosen game engine. Any limitations in integrating Python libraries with Unreal Engine or Unity could impact the design and AI features.

• System Requirements and Hardware Constraints: The game will be developed for mid-range gaming PCs, with the assumption that players will have a system meeting basic specifications (e.g., 8GB RAM, mid-range GPU). If the hardware capabilities change significantly during development, it may require adjustments in graphics, performance optimization, and the overall game experience.

• Third-Party Software and Tools: The project will use third-party tools for tasks like asset creation, procedural generation, and possibly sound effects. It is assumed that these tools will remain compatible with the chosen engine and that licensing costs will be affordable. Any changes to the availability, pricing, or compatibility of these tools could impact the development process.

• Procedural Generation Requirements: The assumption is that procedural generation will remain feasible and scalable within the constraints of the chosen game engine. If limitations arise in implementing a truly randomized experience, adjustments to the game's replayability and map design may be needed.

• Testing and Debugging Tools: It is assumed that suitable testing and debugging tools will be available and integrated with the development environment. If any tools become unavailable or incompatible, alternative solutions will need to be evaluated to maintain testing standards.

• Console Porting: The assumption is that once the PC version is stable, the game will be ported to consoles. This process will depend on the compatibility of the chosen engine with console platforms and the necessary adjustments for control schemes, performance, and hardware limitations. If console porting proves too complex or incompatible, the scope may be adjusted to focus solely on PC platforms.

• User Base and Feedback: It is assumed that the target user base for Spaceborn will remain consistent with the current market research, with a preference for survival and strategy games. Any significant shift in player expectations or trends may require changes in gameplay mechanics or feature prioritization.

2.6 Apportioning of Requirements

The following features and functionalities are identified as enhancements that may be included in future versions of Spaceborn. These are not critical to the core gameplay and can be delayed for later updates:

• Advanced Machine Learning for Enemy AI: Expanding the machine learning-driven AI to incorporate collaborative strategies among enemies or adapt to multiple playstyles more dynamically.

• Console Porting: Developing versions compatible with major gaming consoles, such as PlayStation, Xbox, and Nintendo Switch. This includes optimizing performance, adjusting controls, and ensuring compliance with platform-specific requirements.

• Game Engine Selection: The final decision between Unreal Engine and Unity will be delayed to a future phase. This allows for an evaluation of each engine's capabilities in terms of graphical fidelity, performance, and integration with machine learning and procedural generation systems.

3. Specific Requirements

This section outlines all the software requirements for the Spaceborn game, providing the necessary details to design, implement, and test the system. These requirements cover the inputs, outputs, and all functions the system must perform to meet the expectations of users and ensure successful gameplay.

3.1 External Interface Requirements

This section describes the inputs and outputs for Spaceborn. It includes data formats, interaction methods, and system responses to external stimuli.

3.1.1 Input and Output Overview:

Name of Item: Player Input (e.g., Movement, Interaction)

• Purpose: To allow players to control their character and interact with the game world.

• Source of Input: Player actions through keyboard/mouse/controller.

• Valid Range: Movement range within the game map (0–1000 units).

• Timing: Input is processed in real-time during gameplay.

• Relationships to Other Inputs/Outputs: Player movement affects the AI behavior, resource collection, and environmental changes.

Name of Item: AI Behavior Update (Mutants, Enemies)

• Purpose: To process AI decisions and update the behaviors of mutants and enemies based on player interactions.

• Source of Input: AI algorithms that analyze player position and actions.

• Output: Mutant movement, attacks, or strategy changes.

• Valid Range: Behavior can range from passive (avoidance) to aggressive (attack).

• Timing: AI behavior updates every 1-2 seconds, triggered by player interactions or environmental changes.

• Relationships to Other Inputs/Outputs: AI behavior is influenced by player actions and environmental conditions.

Name of Item: Save Game Data

• Purpose: To allow players to save and load their game progress.

• Source of Input: Player action (Save command).

• Output: A saved game file, including player inventory, AI states, and resource levels.

• Valid Range: Save game data must include up to 3 game slots for multiple save states.

• Timing: Save operation happens when the player manually saves or at checkpoints.

• Relationships to Other Inputs/Outputs: Saves include data from the current game session, which may include all player interactions and progress.

3.2 Functional Requirements

3.2.1 Save Game Functionality

• Introduction/Purpose of Feature: This feature allows players to save their current progress in Spaceborn, including game state, inventory, and resource levels.

• Stimulus/Response Sequence:

Stimulus: The player presses the "Save" button or triggers the save command from the menu.

Response: The system stores the game state (including player progress, resources, and AI states) and provides a confirmation message, such as "Save Successful."

• Associated Functional Requirements: The system shall save the player’s progress, including inventory, resources, completed mini-games, and AI behaviors. The system shall provide multiple save slots for players to choose from and manage their progress.

3.2.2 AI Behavior Updates

• Introduction/Purpose of Feature: This feature ensures the AI behavior of mutants and enemies is updated based on player interactions and events in the game.

• Stimulus/Response Sequence:

Stimulus: Player actions such as movement, attacking, or interacting with certain game elements (e.g., triggering alarms).

Response: The AI system updates the mutants' behaviors, such as attacking, avoiding, or preparing for a counteraction based on the player's actions.

• Associated Functional Requirements:

The system shall update the AI behavior of mutants in real-time as the player interacts with the environment.

The system shall track and adjust the mutant's AI state based on factors such as health, aggression level, and location.

3.2.3 Resource Management

• Introduction/Purpose of Feature:

This feature manages the collection and consumption of resources (e.g., food, oxygen, energy) within the colony and the spaceship.

• Stimulus/Response Sequence:

Stimulus: The player interacts with the environment, collects, or uses resources.
Response: The system updates the resource count and adjusts the colony’s status accordingly (e.g., when resources are low, the game may prompt the player for action).

• Associated Functional Requirements:

The system shall track and update resource levels, including food, oxygen, and energy.
The system shall alert the player when resources are running low or when certain thresholds are reached. requirements define the specific actions the system must perform in response to inputs and how it should process and generate outputs.

3.2.4 Main Menu Use Case

Brief Description:

The use case diagram outlines a main menu that provides users with several options to engage with the system. The main menu includes the following features: starting a new game, loading a saved game, accessing settings, and exiting the application.

Initial Step by Step Description:

1. New Game:When the player clicks on the "New Game" option, a fresh gaming session begins.

2. Load Game: Upon selecting the "Load Game" option, a list of previously saved game sessions is displayed.
3. Settings: Choosing the "Settings" option brings up a configuration panel for adjusting various game parameters.
4. Exit: Selecting the "Exit" option ends the current session or closes the application.

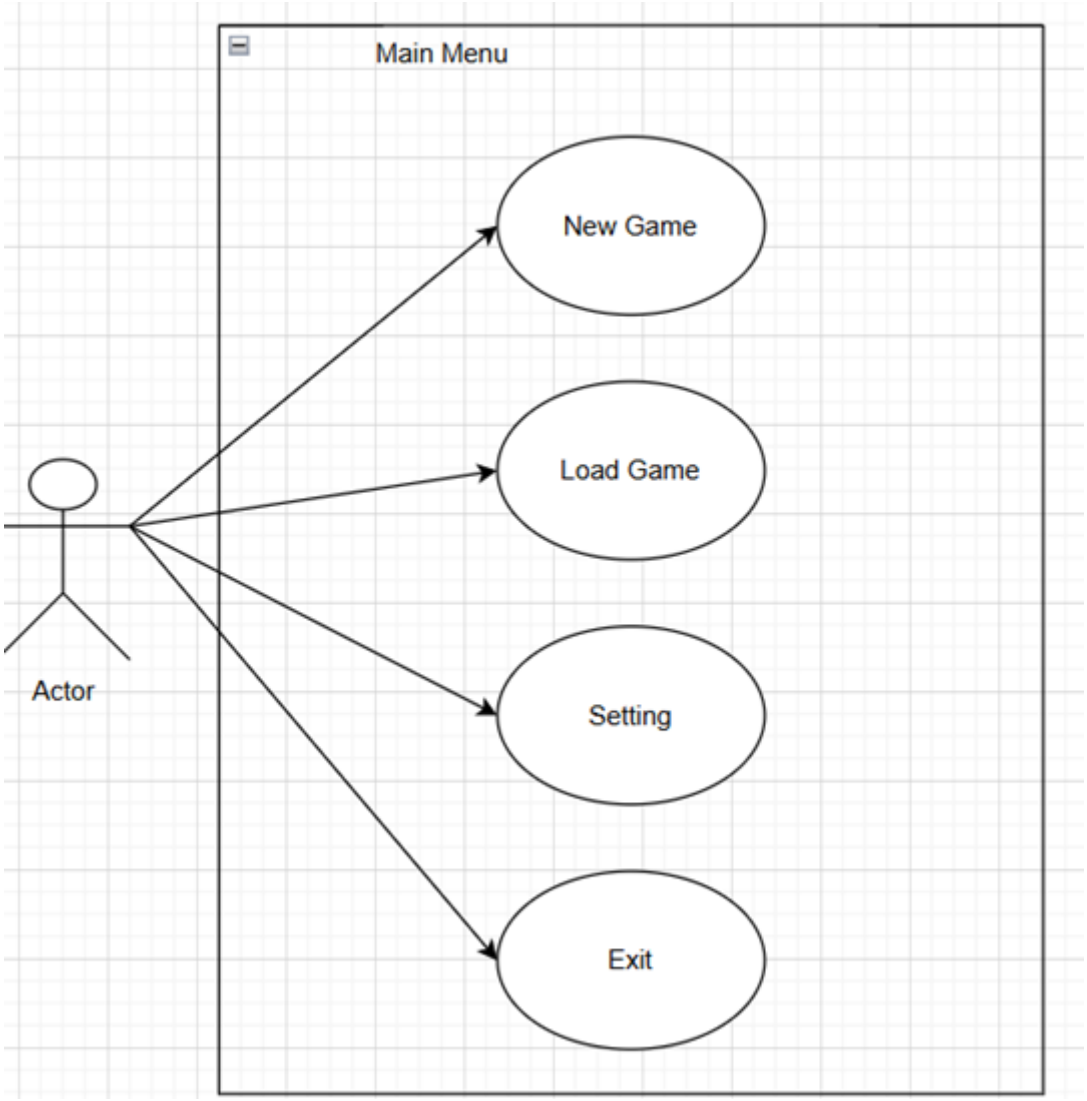


Figure 1: Main Menu Use Case [1]

3.2.5 Gameplay Use Case

3.2.5.1 Brief Description:

Players utilize Resource Data to access essential information about critical resources like oxygen, energy, and food, guiding their strategic actions. Colonist Data provides details on the health, morale, and productivity of each colonist aboard the spaceship. Specialized Systems Data offers insights into the status of vital systems like oxygen generators, power grids, and food production units. The Timeline & Minimap help players track progress, locate areas of interest, and identify threats in the environment. The Stop-Start feature enables players to control the pace of gameplay, pausing during critical decision-making

or real-life interruptions. In-Game Settings offer customization options similar to the main menu, allowing players to adjust graphics, sound, and controls.

Interactive buttons such as Resource Management, Construction, and Defense provide targeted menus for managing resources, building systems, and organizing the spaceship's defense mechanisms. The Abilities Button allows players to activate special abilities, such as boosting colonist morale or temporarily increasing resource production. The Selected Button Option serves as a confirmation interface, showing the player's choices for actions or upgrades before finalizing them.

Additionally, players can engage in mini-games during specific events, such as repairing systems or countering hostile mutant attacks. Successful completion of these mini-games provides strategic advantages, such as temporarily enhanced defenses or bonus resources, helping players stay ahead of challenges.

3.2.5.2 Initial Step-by-Step Description:

Resource Data: Players can review Resource Data to monitor critical resources like oxygen, food, and energy. This allows them to allocate resources efficiently and address shortages.

Colonist Data: Players navigate to the Colonist Data section to access specific information about each colonist's health, morale, and productivity. This helps prioritize medical care or allocate work tasks.

Specialized Systems Data: Users examine Specialized Systems Data to assess the operational status of key spaceship systems, such as oxygen generators, energy grids, and food production units. Malfunctions or inefficiencies are highlighted for repair or upgrades.

Timeline & Minimap: The Timeline & Minimap feature enables players to track in-game progress, identify threats, and explore new areas of the spaceship or nearby objects in space. It ensures players remain aware of the overall game context.

Stop-Start: The Stop-Start functionality allows players to pause or resume gameplay, providing control over the pacing during critical moments, such as system malfunctions or combat scenarios.

In-Game Settings: Accessing the In-Game Settings menu lets players adjust parameters like graphics, sound, and resolution. Save, load, and exit options are also available for flexibility during gameplay.

Resource Management Button: Clicking on the Resource Management Button opens a menu displaying available resources and their usage. Player can reallocate resources to prioritize critical systems during shortages.

Construction Button: Selecting the Construction Button shows a list of structures and systems available for building or upgrading. Information about resource costs and benefits is provided to guide strategic decisions.

Defense Button: The Defense Button allows players to deploy or upgrade defensive systems, such as turrets, traps, or shields, to counter mutant threats. Strategies for covering vulnerable areas are emphasized.

Button Option: The Selected Button Option interface previews all chosen actions, such as resource reallocation, construction plans, or defensive measures. Players can confirm or modify their decisions before finalizing.

Mini-Games: During key events, such as system repairs or mutant attacks, players can engage in mini-games to gain advantages. For example:

• Successfully completing a repair mini-game restores a critical system faster.

• Winning a combat mini-game temporarily boosts defenses or damages attacking mutants.

• Completing an exploration mini-game unlocks bonus resources or rare upgrades.

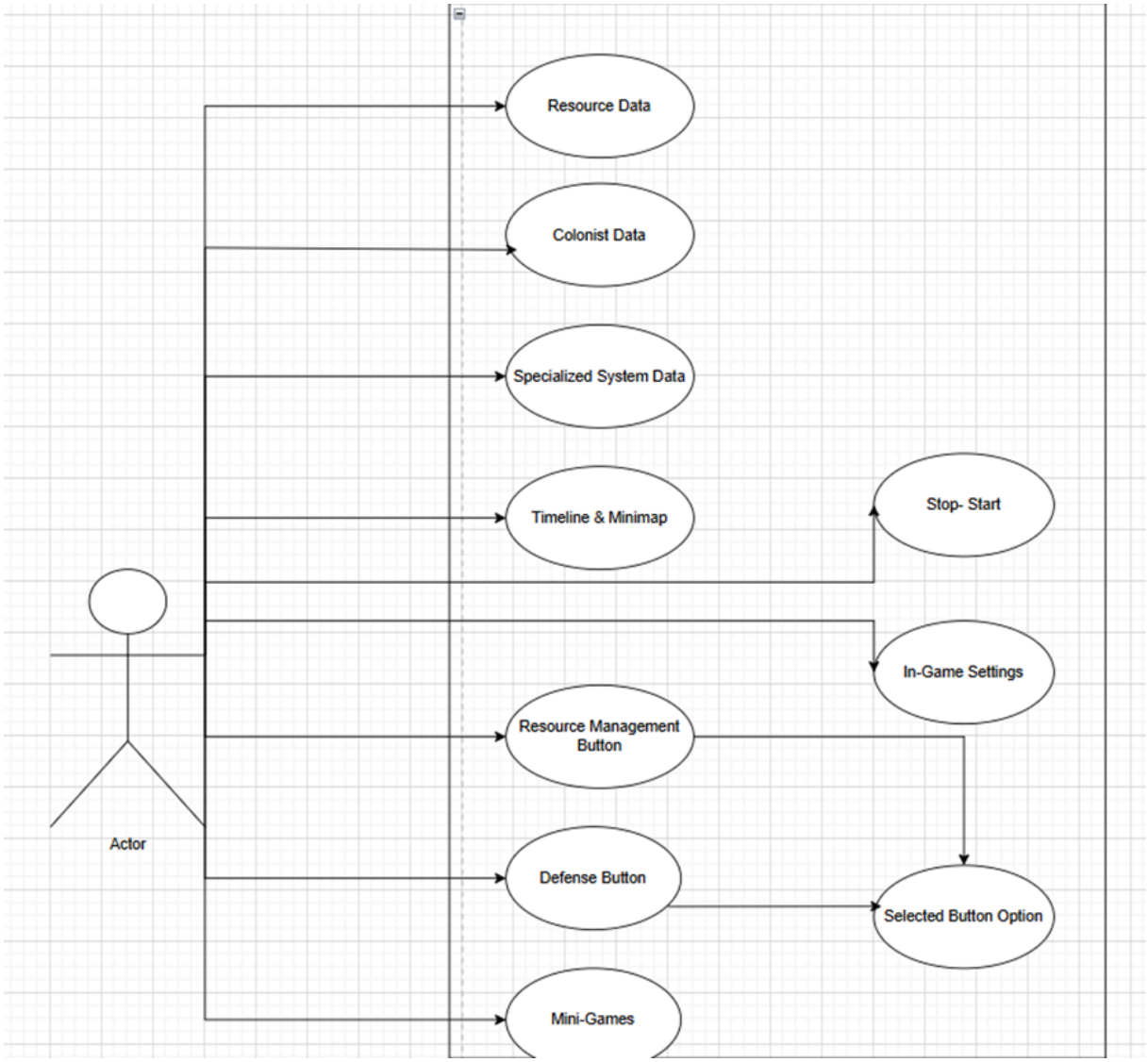


Figure 2: Gameplay Use Case [1]

3.2.6 In-Game Settings Menu Use Case

3.2.6.1 Brief Description

The use case diagram describes a settings menu where player can adjust Music Volume, Resolution, and Display Mode. They also have options to Cancel changes, Save modifications, or Quit the Game, providing a straightforward and user-friendly interface for customizing their preferences within the system.

3.2.6.2 Initial Step-by-Step Description

Music Volume: Player can adjust the background music volume to their preference.

Resolution Button: Player can modify the game's resolution to fit their PC's display capabilities.

Display Mode: Player can toggle between fullscreen and windowed modes for the game.

Cancel Button: This option allows players to discard any changes made during the session.

Save Button: This option saves all adjustments made to the settings.

Quit Game: Selecting this button takes the player back to the main menu, ending the current session.

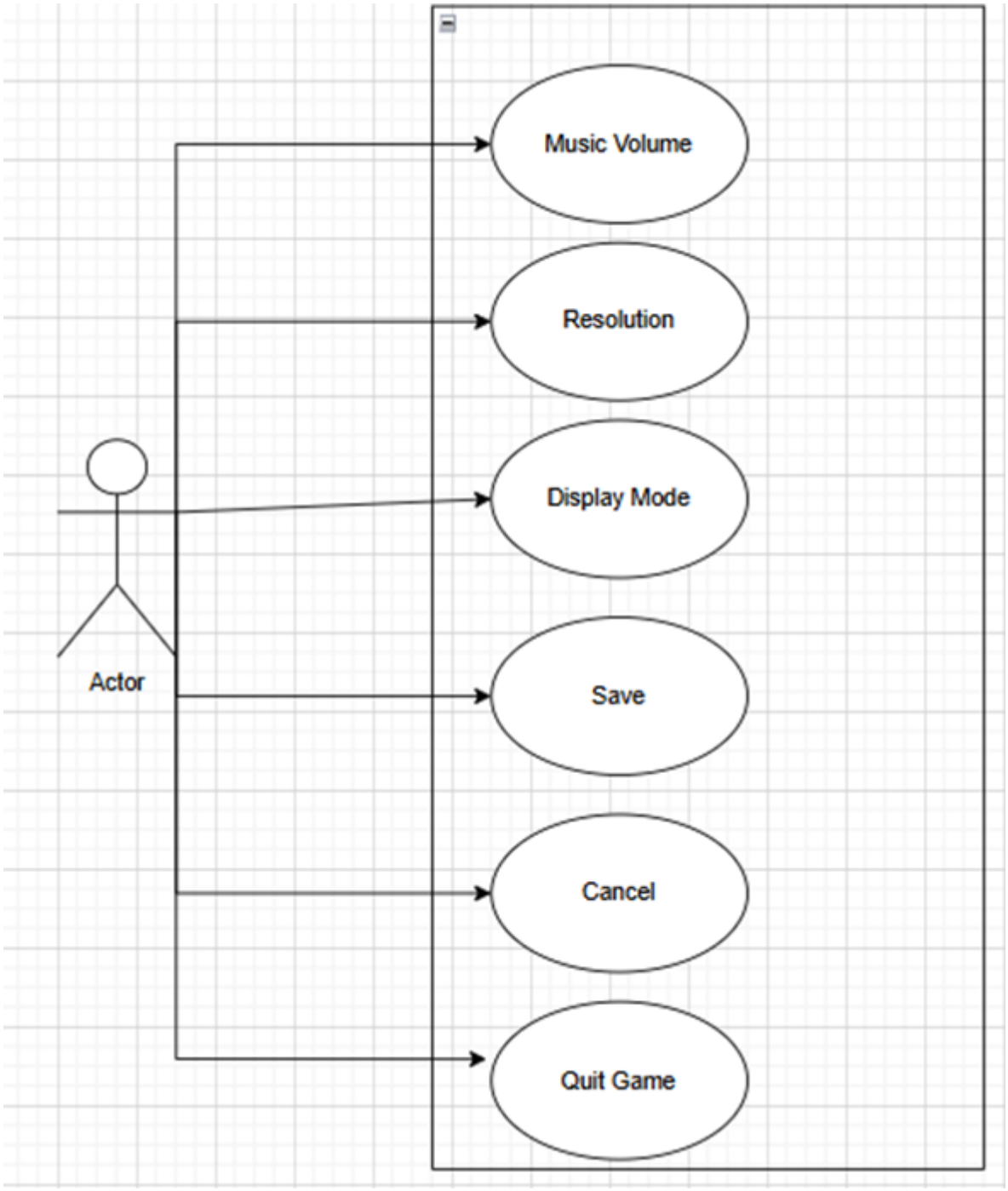


Figure 3: In-Game Settings Menu Use Case [1]

3.3 Error Handling

The system must handle errors gracefully, ensuring that any issues do not disrupt the player's experience.

• The system shall display a meaningful error message if the save file is corrupted or if there is a failure in saving or loading game data.

• The system shall provide recovery options, such as retrying or returning to the main menu in case of save/load failure.

• The system shall ensure that AI behavior remains consistent and error-free even when unexpected player actions occur.

3.4 Performance Requirements

The Spaceborn game should meet the following performance criteria to ensure a smooth gameplay experience:

The system shall load saved game data within 5 seconds.

The system shall update AI behavior for all mutants/enemies in real-time with no noticeable lag or delay.

The system shall handle up to 5 simultaneous AI agents (mutants) without affecting frame rates or gameplay performance.

Smooth and latency-free simulation visuals are crucial for maintaining a high level of immersion, and achieving this depends on various aspects of the user's PC. The minimum requirements include:

OS: Windows 7 or better

Processor: Intel i5 3.1 Ghz Quad Core

Memory: 2 GB RAM

Graphics: Nvidia GTX 660 / Radeon HD 7800 or better

Recommended Display Resolution: 1440x900

3.5 Logical Database Requirements

The Spaceborn game will include a database to store key information required for gameplay, focusing on saved game data and AI behavior. The following logical database requirements apply:

Types of Information Used by Various Functions:

• Save Game Data: Player progress, inventory, completed mini-games, and unlocked resources.

• AI Behavior Data: Information on mutant/alien types, their locations, and behavior patterns.

Frequency of Use:

• Save Game Data: Accessed during save/load operations to store and retrieve player progress.

• AI Behavior Data: Accessed during gameplay to update and retrieve enemy locations and behavior decisions.

Accessing Capabilities:

• Read: Used to load saved game data and AI behavior states.

• Write: Used to update player progress,and modify AI behavior patterns as the game progresses.

Data Entities:

• Player (ID, Name, Progress)

• Mutants/Enemies (ID, Type, Location, Behavior)

Relationships:

• Player ↔ Save Game: A player's progress is saved and loaded between sessions.

• Mutants ↔ AI Behavior: Mutants have specific behaviors stored in the database based on game progress.

Integrity Constraints:

• Data Integrity: Save game data must be accurate and consistent between sessions.

• AI Behavior Integrity: Mutant/alien behavior data must align with game progression and previous interactions.

Data Retention Requirements:

• Saved Games: Retained between sessions, with each saved game lasting at least one year from the last gameplay session.

• AI Behavior Data: Retained throughout the session and adjusted based on player actions.

3.6 Design constraints

The following design constraints will be imposed on the Spaceborn project based on external standards, hardware limitations, and gameplay requirements:

• Game Engine and Machine Learning Constraints: The choice of game engine (Unreal Engine or Unity) will dictate certain design decisions, including rendering techniques, AI behavior implementation, and memory management. The machine learning models used for AI (such as for mutant behavior) must be compatible with the selected game engine, which could restrict certain algorithms or performance features.

• Hardware Limitations: The game must be optimized to run efficiently on mid-range gaming PCs. This includes managing memory usage, optimizing frame rates, and ensuring

smooth gameplay on typical consumer-grade hardware. The game must also be scalable for possible future console ports, which could introduce additional hardware constraints, such as optimizing for lower RAM and CPU capacities.

3.6.1 Standards compliance

While Spaceborn will not be subject to specific industry regulatory standards, the following general compliance and best practices will be followed:

• Data Integrity Standards: The game must follow basic data integrity standards to ensure that saved games and player progress are preserved between sessions. Backup mechanisms and error handling for corrupted save files will be implemented.

• Privacy Standards for Player Data: Any player data (e.g., saved games) will be stored and handled securely, following industry standards for privacy and data protection.

• Accessibility Standards: The game will incorporate accessibility features such as customizable key bindings, colorblind modes, and subtitle options to ensure it is accessible to a wide audience.

• Logging and Debugging Standards: During development, logging standards will be followed to track errors and gameplay issues, ensuring that bugs are logged and traced back for resolution. These logs will be stored securely and not include any personally identifiable information.

3.7 Software System Attributes

3.7.1 Reliability

The reliability of the "Spaceborn" will be established through:

• Rigorous testing of mini-games (e.g., puzzles for oxygen generation, energy restoration) to ensure they function correctly under all scenarios.

• Stress testing the procedural map generation and alien behavior to guarantee stable performance across varying gameplay conditions.

• Fail-safe mechanisms for critical components, such as preserving embryonic status even during unexpected disruptions.

• Backup systems to restore the latest gameplay state in case of crashes.

3.7.2 Availability

This To ensure high availability for "Spaceborn":

• Checkpoint System: Autosave functionality will activate at key milestones, such as after solving a mini-game or escaping alien encounters.

• Recovery: If the game unexpectedly shuts down, the recovery system will reload the most recent checkpoint seamlessly.

• Restart Capability: Allow individual systems (e.g., machine learning-driven AI modules) to restart independently in case of failure, preventing complete game downtime.

3.7.3 Security

To ensure high availability for "Spaceborn":

• Checkpoint System: Autosave functionality will activate at key milestones, such as after solving a mini-game or escaping alien encounters.

• Recovery: If the game unexpectedly shuts down, the recovery system will reload the most recent checkpoint seamlessly.

• Restart Capability: Allow individual systems (e.g., machine learning-driven AI modules) to restart independently in case of failure, preventing complete game downtime.

3.7.4 Maintainability

Spaceborn will be designed for ease of maintenance through:

• Modular Architecture: Each system(mini games, procedural map generation, and AI behavior) will function as independent modules to simplify debugging and updates.

• Documentation: Comprehensive documentation for the C++, Python, and Unreal Engine components will ensure maintainability.

• Standardized Coding Practices: Adhere to best practices in both C++ and Python, utilizing design patterns and consistent naming conventions.

• Version Control: Use GitHub for source code versioning, enabling efficient rollback and collaborative development.

3.7.5 Portability

The portability of "Spaceborn" will ensure performance on a range of devices: • Proven Tools: Utilize Unreal Engine and its cross-platform capabilities for seamless deployment on PC and future console support.

• Language Choices: Use C++ and Python for platform-independent development.

• Optimization: Prioritize optimization techniques, such as efficient asset loading and lightweight AI computations, to ensure smooth performance even on mid-range PCs.

Software Design Description (SDD)

1.Introduction

1.1 Purpose:

This document aims to provide a comprehensive overview of the "Spaceborn" project's software design, ensuring clarity on the system's architecture, technical components, and data flow. It serves as a technical guide for developers, testers, and stakeholders to understand the design choices, implementation details, and integration points across the system.

1.2 Definitions:

The Spaceborn project is a space-themed survival and colony management game where players manage resources, build systems, and protect their colony from external threats. The game incorporates procedural map generation, AI-driven characters, and dynamic events. The scope includes:

- Procedural Map and Event Generation
- AI for NPC Behavior and Combat Systems
- Resource and Colony Management
- Immersive Survival Gameplay with Strategic Elements.

1.3 Overview

This document covers the following areas:

- System Architecture: Describes the software layers and component interactions.
- Data Management Strategies: Defines database design, data security, and backup policies.
- Hardware and Software Requirements: Details system prerequisites for optimal performance.
- System Interfaces: Describes how users and other systems interact with Spaceborn.
- Scalability and Performance Requirements: Ensures the system can grow and meet performance benchmarks.
- Security Protocols: Protects user data and maintains secure communication.

1.4 Glossary

Term	Definition
UI:	User Interface
AI:	Artificial Intelligence
FPS:	Frames Per Second
HUD	Heads-Up Display

2. System Overview:

2.1 System Objectives

The primary objectives of the Spaceborn system include:

- Creating an immersive colony-building and survival experience.
- Utilizing procedural generation to ensure replayability.
- Integrating AI-driven NPC behavior and events.
- Ensuring high performance across multiple platforms (PC, VR, consoles).

2.2 System Architecture

Spaceborn follows a layered architecture, dividing functionality into the following layers:

1. Presentation Layer: User interface elements, HUD, and menu systems.
2. Application Layer: Game logic, resource management, combat systems, AI behavior.
3. Data Layer: Database interactions, game state storage, and inventory management.
4. Networking Layer (Future Scope): Multiplayer synchronization and communication.

3. Data Design

3.1 Database Design

The game uses a relational database system to manage persistent and transient data.

Key Tables:

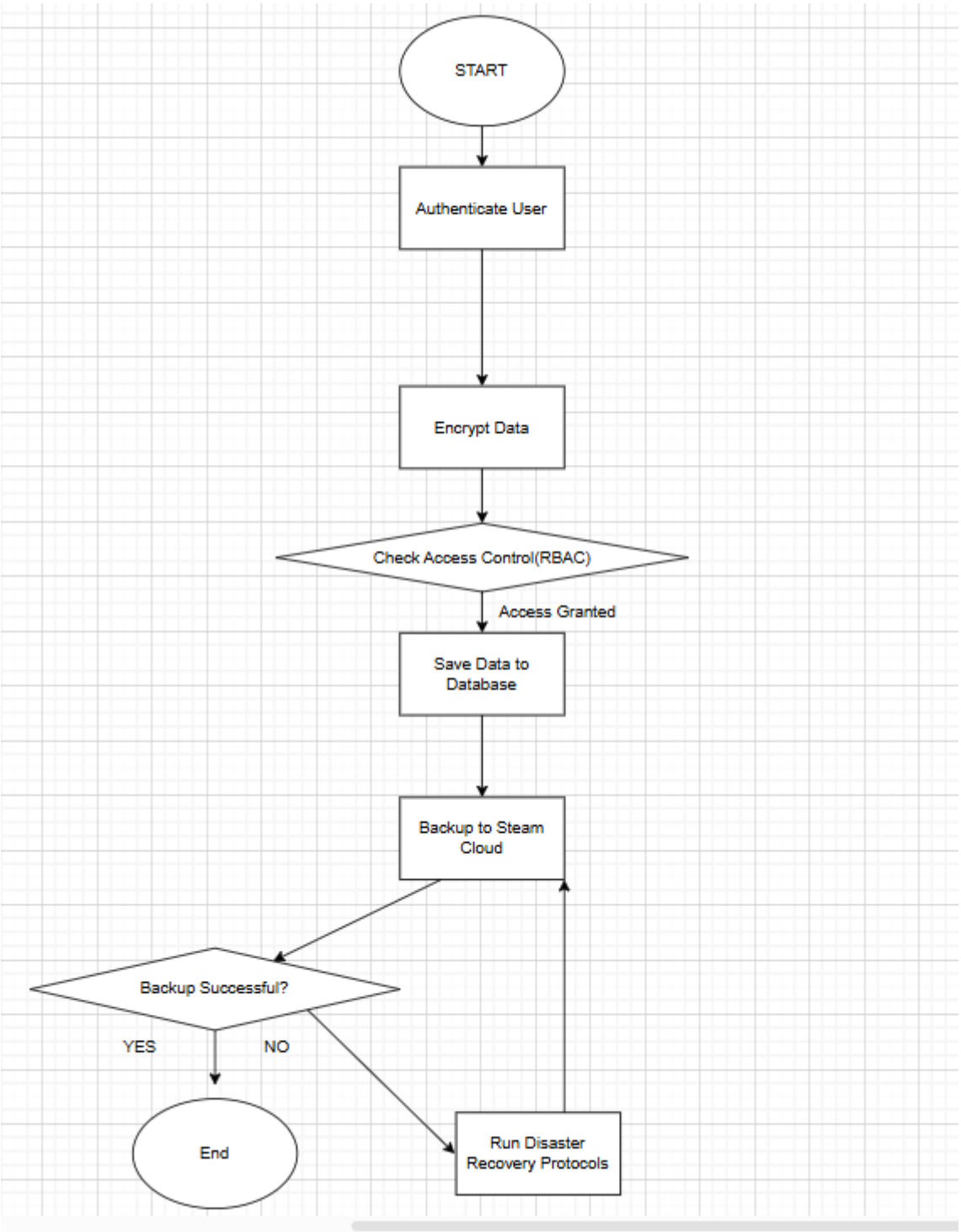
- Users: Stores player profiles, credentials, achievements.
- Inventory: Tracks collected resources and crafted items.
- GameState: Records checkpoints, mission progress, and environmental states.
- NPCState: Manages NPC health, status, and behavior triggers.

3.2 Data Security

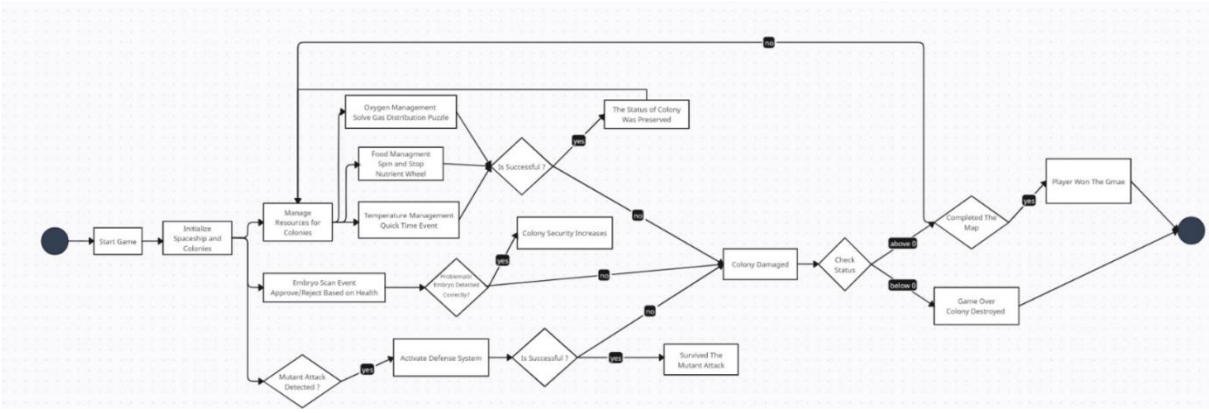
- Authorization and Authentication: Only authorized persons can access player data.
- Backup: Automated cloud backups using Steam Cloud API.
- Access Control: RBAC (Role-Based Access Control) to limit unauthorized access.

3.3 Data Backup and Recovery

- Incremental data backups reduce resource load.
- Disaster recovery protocols ensure minimal data loss in failures.

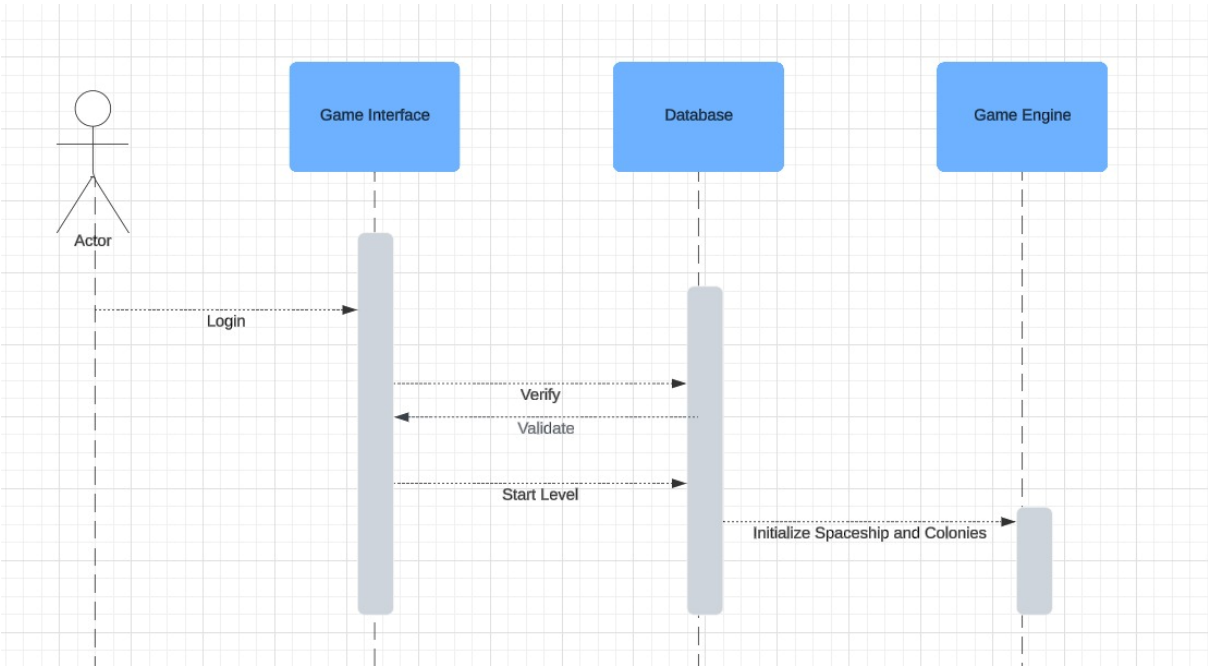

START
Authenticate User
Encrypt Data
Check Access Control(RBAC)
Access Granted
Save Data to Database
Backup to Steam Cloud
Backup Successful?
YES
NO
End
Run Disaster Recovery Protocols

3.4 Activity Diagram

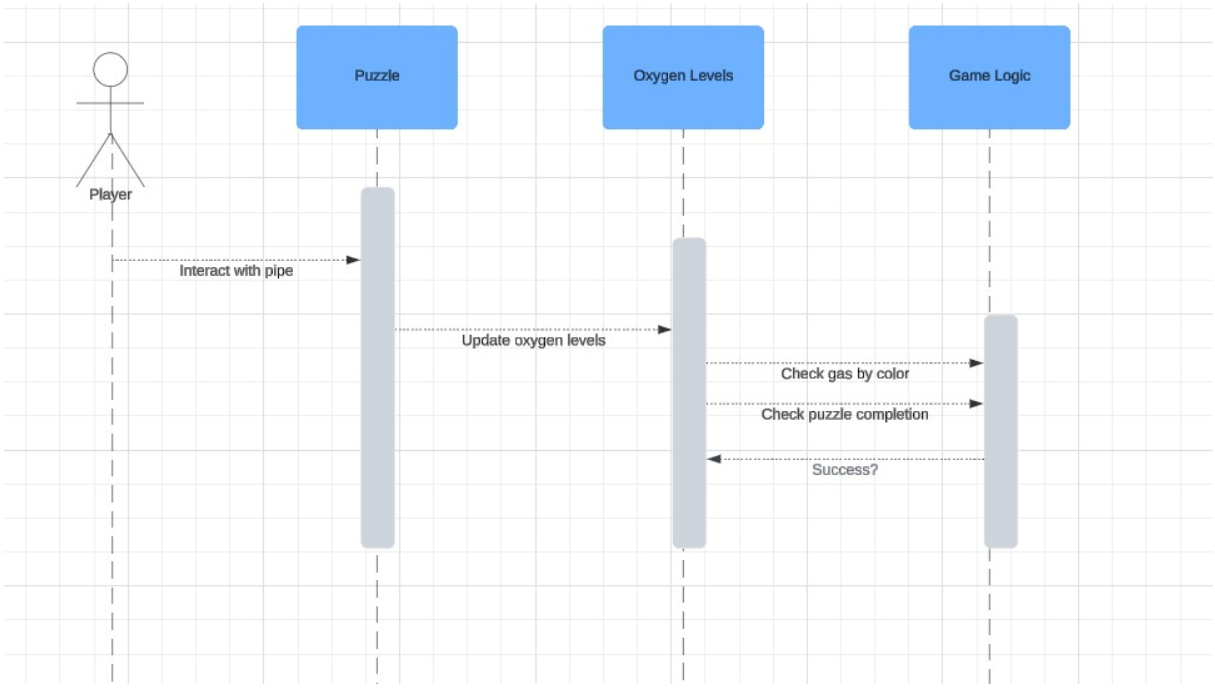


3.5 Sequence Diagram

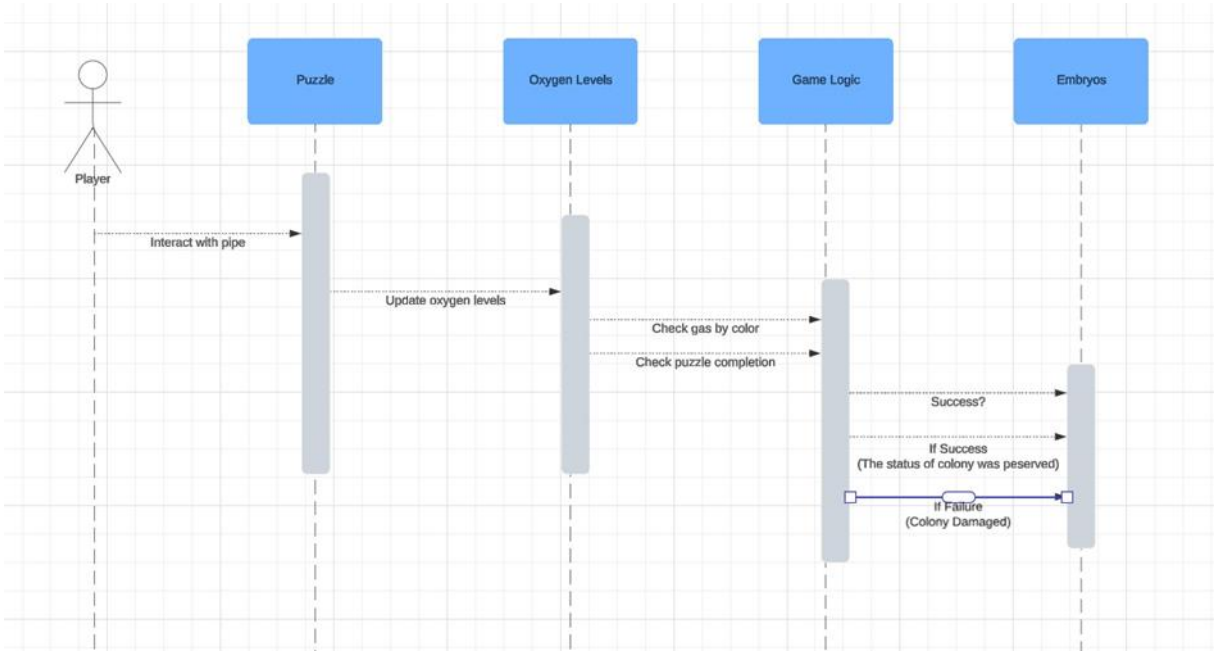
3.5.1 Login and Initialize Game



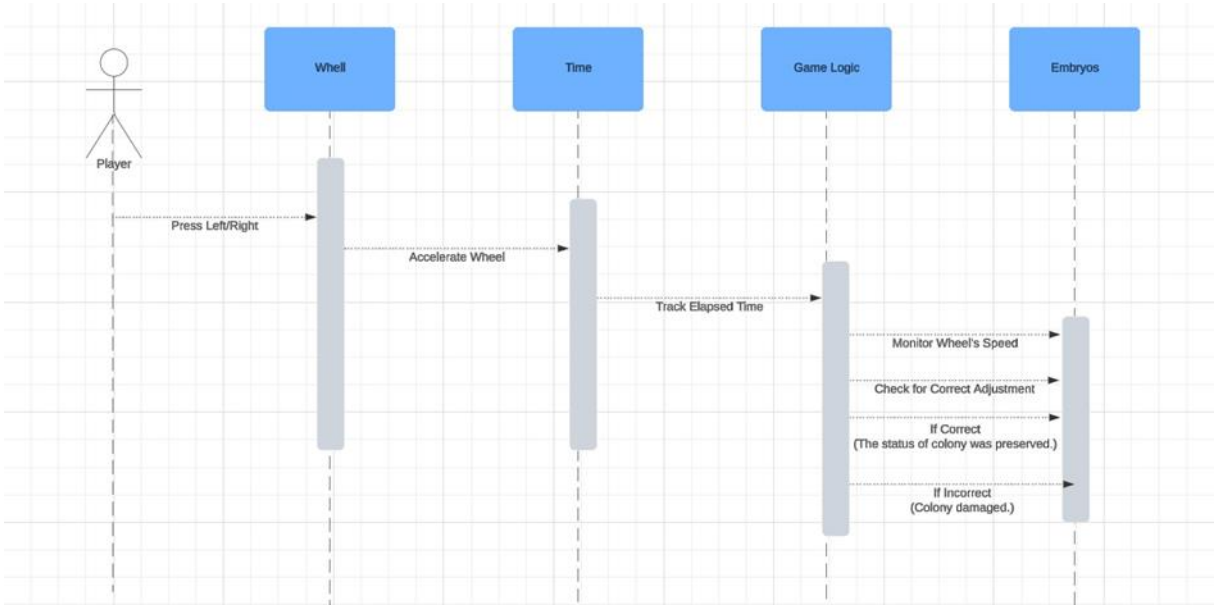
3.5.2 Oxygen Management Puzzle



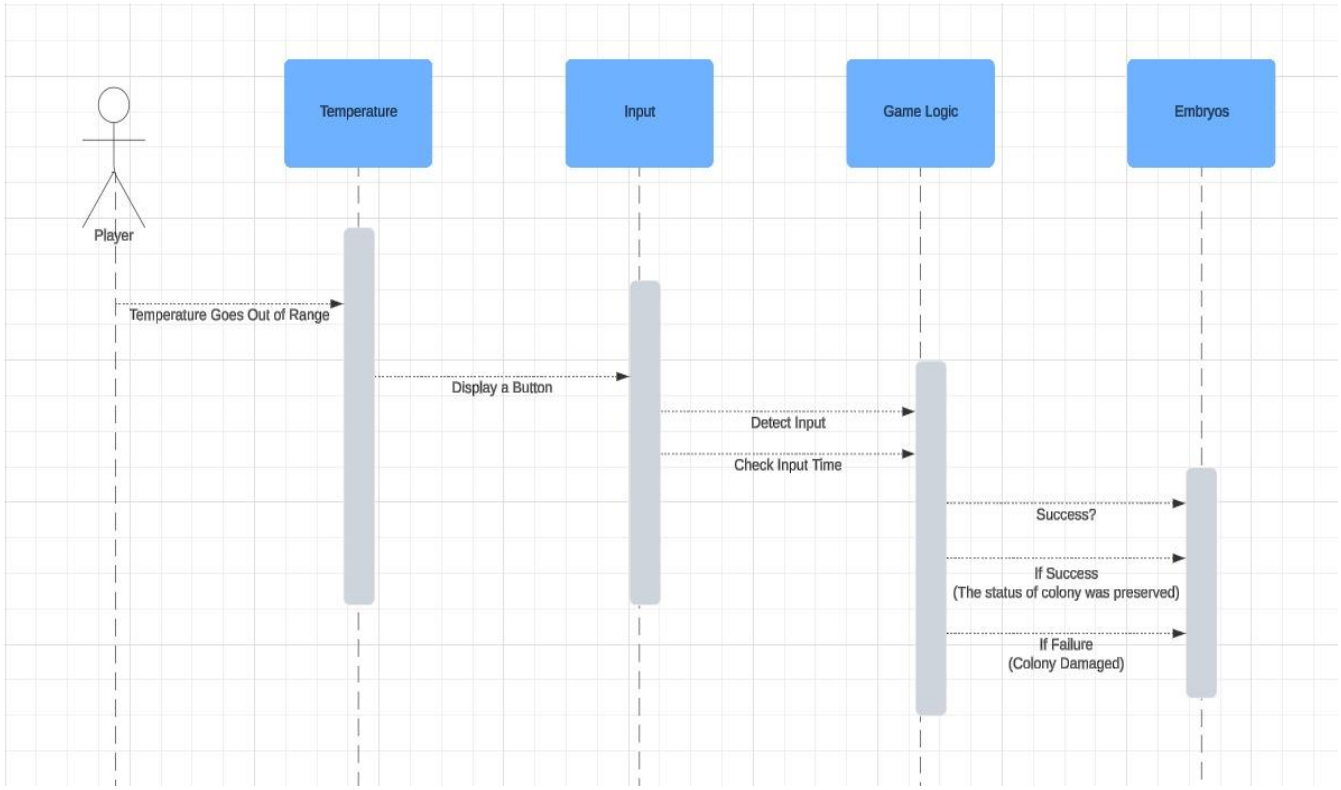
3.5.3 Oxygen Management and Colony Status



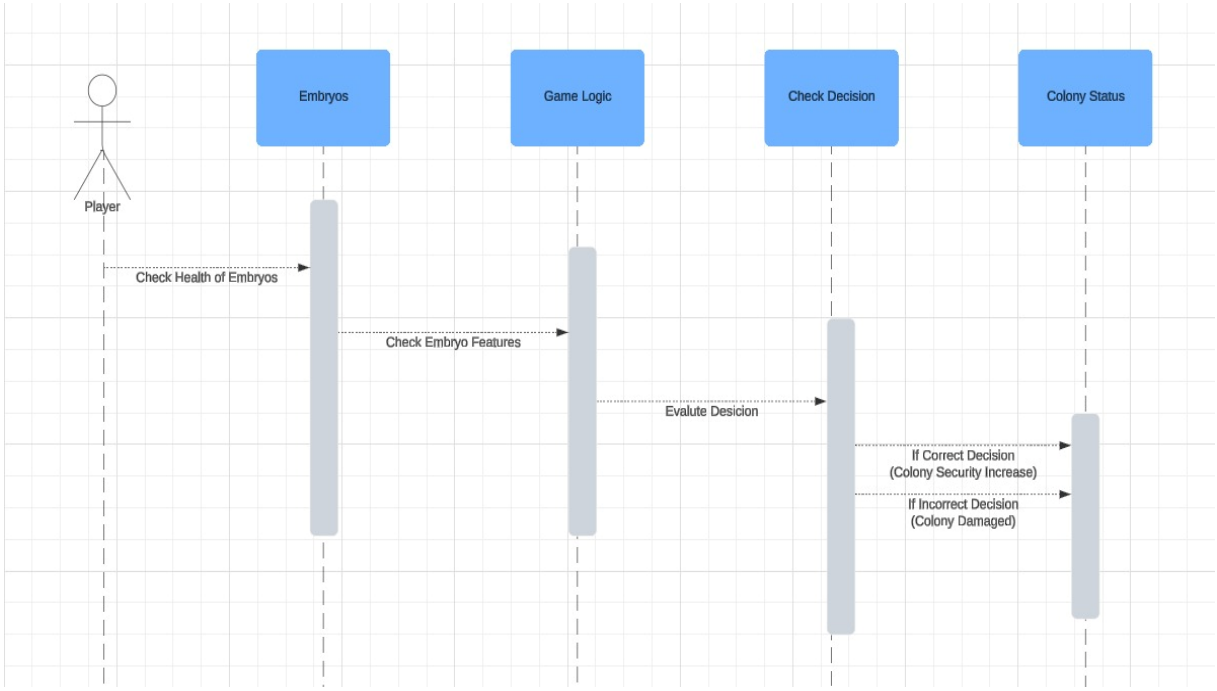
3.5.4 Food Management Wheel



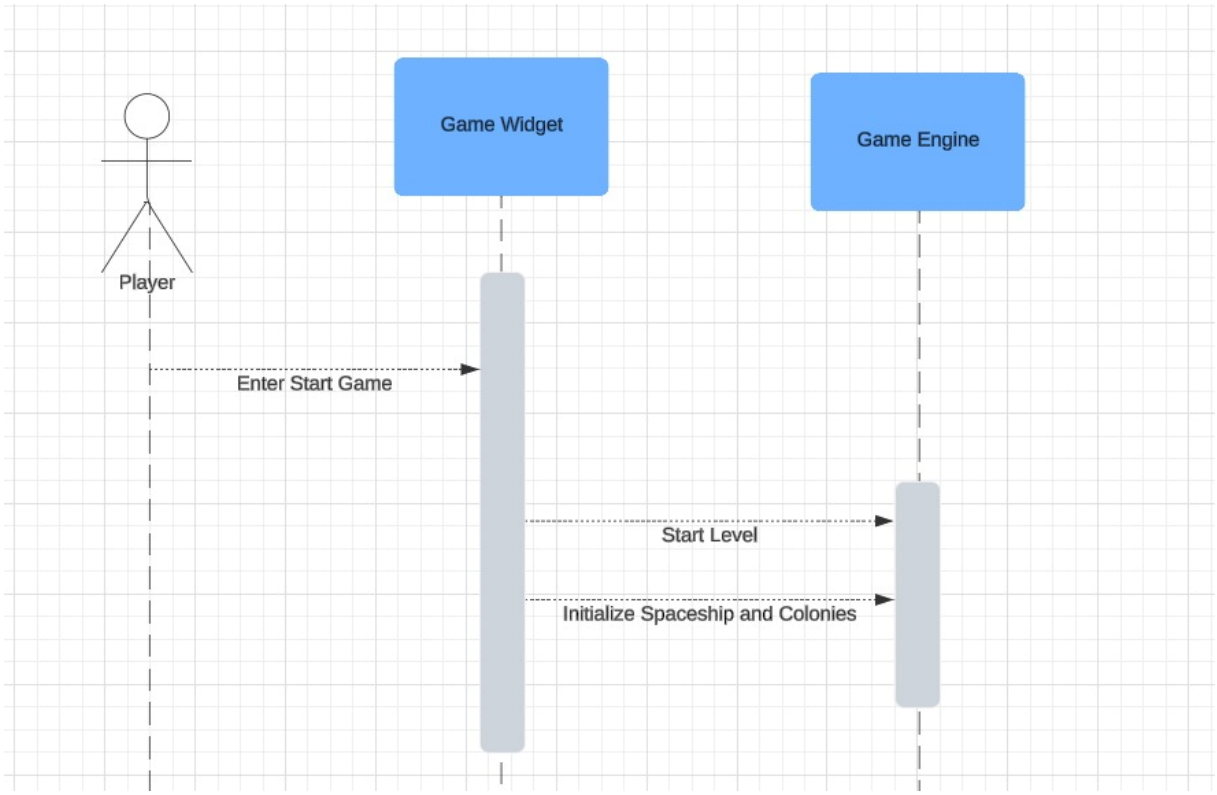
3.5.5 Temperature Management



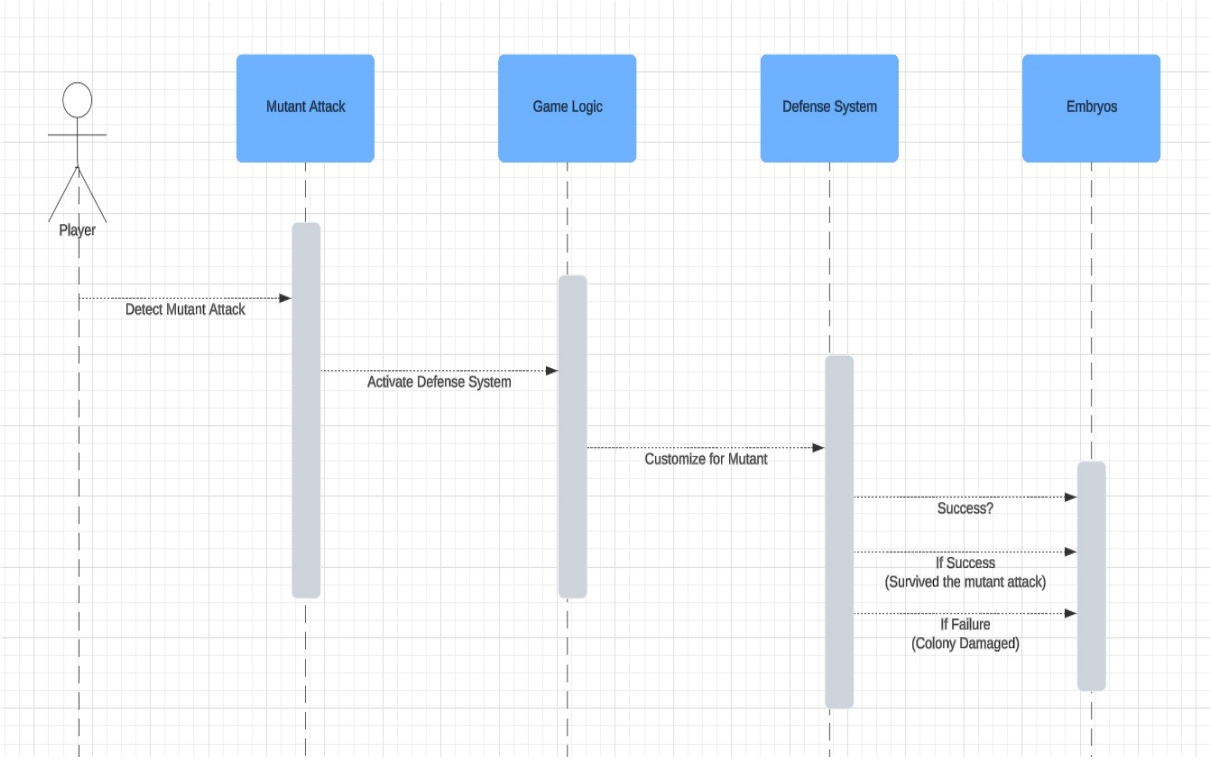
3.5.6 Embryo Health Check



3.5.7 Start Game Sequence



3.5.8 Mutant Attack Scenario



4. Interface Design

This section describes the basic interfaces of Spaceborn and the purpose of each. The interfaces are designed to allow players to log in to the game, manage their colonies, and view important information.

Login Page

Description:

This is the page where players perform their identity verification.

Username and password are entered.

Player progress, achievements and game preferences are loaded from the database.

Key Features:

Username and password input boxes.

"Log In" and "Forgot My Password" buttons.

Game version and update notes information.

Purpose: To provide players with safe and fast access to the game.

Simulation Start Page:

Description:

This is the page where the player makes the necessary settings to start the spaceship and colonies.

Players can choose the game difficulty level and starting resources.

Colony and spaceship systems are started here.

Basic Features:

Difficulty level selection (Easy, Medium, Hard).

Editing starting resources (Oxygen, Food, Energy).

"Start" button.

Purpose: To prepare the player for the game by making strategic starting settings.

Simulation View Page

Description:

This is the page where the player makes the necessary settings to start the spaceship and colonies.

Players can choose the game difficulty level and starting resources.

Colony and spaceship systems are started here.

Basic Features:

Editing starting resources (Oxygen, Food, Energy).

"Start" button.

Purpose: To prepare the player for the game by making strategic starting settings.

Point Cloud View Page

Description:

This is the page where the environmental data scanned by the spaceship's sensors is visualized for the players.

The data obtained during exploration is shown here in the form of point clouds.

The player can analyze threats or undiscovered resources on this page.

Key Features:

3D Point Cloud Map.

Zoom and Pan options.

Threat and resource markers.

Purpose: To enable the player to make strategic decisions by analyzing exploration data.

References

The resources used to produce this document should be listed here and referenced in the relevant text of this document

[1] Software Engineeering Department, "Graduation Projects," SENG, [Online]. Available: https://seng.cankaya.edu.tr/graduation-projects/. [Accessed 28 June 2024].

https://bilgipaketi.cankaya.edu.tr/CourseInfo?Id=105857&BolumKodu=701&MNo=387

https://catnessgames.com/blog/unreal-engine-5-requirements/

INTRODUCTION TEST PLAN

Version Control

Version No	Description of Changes	Date
1.0	First Version	TBD

Overview

Spaceborn is a space exploration and combat game developed using Unreal Engine 5.4 and hosted on GitHub. The game involves procedurally generated planets, enemy AI, hostage rescue missions, and mini-games aboard the player's ship. The testing process will ensure core mechanics function correctly, including procedural generation, AI behavior, combat mechanics, and ship systems.

Scope

The test plan covers major game mechanics including:

- Procedural planet generation
- AI enemy behavior
- Hostage rescue missions
- Spacecraft controls and movement
- Mini-games on the spaceship
- User interface and interactions

Excluded from this test plan:

- Graphical performance optimizations
- Multiplayer mechanics (if applicable in future updates)
- Advanced physics simulations beyond core gameplay

Terminology

Acronym	Definition
AI	Artificial Intelligence
UE5.4	Unreal Engine 5.4
PCG	Procedural Content Generation
NPC	Non-Playable Character

FEATURES TO BE TESTED

2.1 Procedural Planet Generation (PPG)

- Ensure planets generate unique terrain each time
- Validate correct placement of hostages and enemies

2.2 AI Enemy Behavior (AI-EB)

- AI should detect the player within a specified range
- Enemies should attack when the player is in proximity
- AI should navigate terrain properly

2.3 Hostage Rescue Missions (HRM)

- Hostages should be interactable
- Missions should be marked as complete upon successful rescue

2.4 Spacecraft Controls (SCM)

- Ensure smooth movement in space
- Validate collision detection with planets and objects

2.5 Mini-Games (MG)

- Ensure all mini-games function correctly and provide rewards

2.6 User Interface (UI)

- Menus should be accessible and functional
- Mission objectives should be displayed correctly

FEATURES NOT TO BE TESTED

- Graphics optimization for different hardware setups
- Audio balancing and soundtrack testing
- Future planned multiplayer features

ITEM PASS/FAIL CRITERIA

Criteria	Requirement
Pass	The feature works as intended with no game-breaking issues.
Fail	The feature does not function correctly or causes crashes.

Exit Criteria

- 100% of critical test cases executed

- 95% of test cases pass
- All high-priority test cases pass

TEST RESULT

1. Test Objectives

The purpose of testing the Spaceborn project is to ensure that all game functionalities—including gameplay mechanics, AI behaviors, UI elements, save/load operations, and system performance—meet the requirements defined in the SRS and operate reliably across target platforms.

2. Test Environment

- **Game Engine:** Unreal Engine 5.4
- **Programming Languages:** C++, Blueprint, Python (for ML)
- **Test Systems:**
    - **System A:** Windows 10, Intel i5-9600K, 16GB RAM, NVIDIA GTX 1660
- **Testing Tools:** Unreal Engine Debugger, In-game Console.

3. Tested Features

Feature	Test Case Description	Expected Result	Result
Save/Load System	Save current progress (inventory, AI state, completed objectives) and load it after restarting the game	Game resumes from exact save state	Passed
Procedural Map Generation	Start a new game multiple times to test whether unique layouts are generated every session	Distinct spaceship map generated each time	Passed
Resource Mini-Games	Interact with oxygen, food, energy mini-games and complete them	Game correctly adjusts resource levels	Passed
Machine Learning Enemy Behavior	Allow AI enemy to encounter player repeatedly and observe adaptive behavior	Enemy becomes more efficient in tracking player movements	Passed
In-Game Menu Navigation	Navigate menus: New Game, Load Game, Settings, Exit	Buttons function correctly and redirect appropriately	Passed

Feature	Test Case Description	Expected Result	Result
UI/UX Responsiveness	Click on in-game UI buttons for construction/defense/resource management	Relevant menu opens; action is executed	Passed
Colonist Status System	Monitor colonist morale, health, and productivity	Colonist data is updated dynamically	Passed
Combat Mechanics	Deploy turrets or traps, engage mutant enemies	Mutants are damaged and react accordingly	Passed
Error Handling - Corrupted Save	Attempt to load a manually corrupted save file	System shows error and returns to main menu	Passed
Performance Under Load	Spawn 5 AI enemies simultaneously and trigger events	Game runs at >30 FPS without input lag	Passed
Autosave System	Complete key mini-games and check whether the game autosaves	Autosave file is generated; resumes from correct point	Passed
Settings Menu Functionality	Change resolution, volume, display mode, then save settings	Settings are retained and applied correctly	Passed
Accessibility Features	Enable colorblind mode and test high contrast UI	UI adapts accordingly	Passed
Point Cloud View Functionality	Trigger exploration to display point cloud data	Data visualized in 3D; Zoom and Pan features operational	Passed

4. Bug Reports & Fixes

Bug ID	Description	Status	Resolution
B-101	Save slot 3 failed to overwrite old file	Fixed	Added overwrite confirmation dialog
B-102	AI enemy getting stuck near closed doors	Fixed	Improved navigation mesh generation
B-103	Volume settings not persisting between sessions	Fixed	Linked UI changes to persistent config file
B-104	Mini-map not updating after certain events	Fixed	Added real-time map state refresh
B-105	Embryo status UI overlapping on low resolutions	Fixed	Adjusted HUD scaling and anchors

5. Conclusion

All core game systems have passed functional, usability, and performance testing. Identified bugs were resolved, and the game meets all defined requirements in the SRS. Additional improvements were implemented during development to enhance the overall quality and stability of *Spaceborn*.

REFERENCES

https://github.com/CankayaUniversity/ceng-407-408-2024-2025-Spaceborn/wiki

TEST DESIGN SPECIFICATIONS

6.1

Procedural Planet Generation (PPG)

- Subfeatures: Terrain formation, object placement
- Test Cases: Ensure terrain is unique and objects are positioned correctly

AI Enemy Behavior (AI-EB)

- Subfeatures: Enemy detection, attack behavior, navigation
- Test Cases: AI detects player correctly, attacks, and moves properly

Hostage Rescue Missions (HRM)

- Subfeatures: Hostage interaction, mission completion
- Test Cases: Hostages can be interacted with, mission completes successfully

Spacecraft Controls (SCM)

- Subfeatures: Movement, collision detection
- Test Cases: Ship responds to player inputs and detects collisions

Mini-Games (MG)

- Subfeatures: Functionality, rewards
- Test Cases: Mini-games work and provide intended rewards

User Interface (UI)

- Subfeatures: Menu navigation, mission tracking
- Test Cases: UI elements display correctly and update as needed

6.1 Test Cases

Here list all the related test cases for this feature

TC ID	Requirements	Priority	Scenario Description

PPG.01	Procedural Generation	*High*	Ensure a new terrain is generated each time
AI-EB.01	AI Behavior	H	Enemy detects player within 10m range
HRM.01	Hostage Rescue	H	Hostage responds to player interaction
MG.01	Mini-Games	M	Successfully complete mini-game and receive reward
UI.01	User Interface	L	Mission objectives update correctly

LG.AD.01

TC_ID	LG.AD.01
Purpose	Enter a valid admin user id and password
Requirements	3.1
Priority	High.
Estimated Time Needed	5 Minutes
Dependency	Add User test cases should pass
Setup	An admin user should be created.
Procedure	[A01] Go to login page.
	[A02] Enter a valid admin user id.
	[A03] Enter the valid password for this user
	[A04] Click on the “Login” button.
	[V01] Observe that the login is successful and the admin page appears
	-
Cleanup	Logout

LG.AD.02

TC_ID	LG.AD.02
Purpose	Verify that an appropriate error message is displayed when an invalid admin username or password is entered during login.
Requirements	3.2
Priority	High.
Estimated Time Needed	5 Minutes
Dependency	The LG.AD.01 test case must be successfully completed.
Setup	An admin user with an invalid username or password should be created..
Procedure	[A01] Go to login page.
	[A02] Enter an invalid admin username.

	[A03] Enter an invalid password.
	[A04] Click the “Login” button.
	[V01] Verify that an "Invalid username or password" error message is displayed.
	[V02] The user should not be able to log in and should not be redirected to the admin page.
Cleanup	Logout

6.2 Add User (AU)

6.2.1 Subfeatures to be tested (Add User)

The subfeatures to be tested for the "Add User (AU)" functionality could be:

1. **User Information Input**
   The ability to correctly input the user's first name, last name, email address, password, and other necessary details.
2. **Password Validation**
   Ensuring the password follows the correct format and adheres to password validation rules (length, special characters, etc.).
3. **User Creation**
   Verifying that the process of creating a new user is successfully completed.

4. **User Role Assignment**
   Ensuring that a specific role (admin, user, etc.) is correctly assigned to the user.
5. **Error Handling when Adding User**
   Verifying that appropriate error messages are displayed when incorrect data is entered

7.Detailed Test Cases

- **PPG.01 - Procedural Map Generation**

Ensures that each new game session generates a unique map. Resources, obstacles, and spawn points should be placed logically.

- **IMM.01 - Immersive Story**

Verifies that the game's story progresses according to player choices. Dialogues, voice acting, and animations should be synchronized and engaging.

- **FUN.01 - Fun Gameplay**

Tests whether the game remains enjoyable, maintains a balanced difficulty, and provides rewarding mechanics. Measures player engagement and motivation.

- **UI.01 - Easy to Adapt User Interface**

Ensures that the game's menus and UI elements are intuitive and easy to navigate. Controls should be user-friendly and accessible.

- **MAP.01 - Optimized Map Creation Phase**

Checks if maps are generated quickly and without errors. FPS drops, loading times, and rendering issues are observed.

- **MG.01 - Various Mini-Games**

Ensures that mini-games function correctly and provide the intended rewards. Verifies that player progress is properly recorded upon completion.

- **AI.01 - Smart AI Behaviors**

Tests whether AI reacts logically to player movements and the environment. AI should display behaviors like attacking, evading, and hiding appropriately.

- **FIGHT.01 - Player and AI Combat**

Verifies that combat mechanics are smooth and balanced. Tests player attacks, evasion mechanics, and AI combat intelligence.

7.1 PPG.01 - Procedural Map Generation

TC_ID	PPG.01
Purpose	Ensure that each generated map is unique and contains diverse elements.
Requirements	6.1
Priority	High.
Estimated Time Needed	10 Minutes
Dependency	The procedural map generation system should be implemented.
Setup	The game should be able to generate maps dynamically.
Procedure	[A01] Start a new game session.
	[A02] Generate a new map.
	[A03] Observe terrain features (hills, water, forests, etc.).
	[A04] Restart the game and regenerate a new map.
	[V01] Each new map should have a unique layout.
	[V02] Ensure that essential elements (resources, obstacles, spawn points) are placed logically.
Cleanup	Exit the generated world.

7.2 IMM.01 - Immersive Story

TC_ID	IMM.01
Purpose	Ensure that story elements are delivered correctly and interactively.
Requirements	6.2
Priority	High.
Estimated Time Needed	8 Minutes
Dependency	Storytelling mechanics should be implemented.
Setup	The game should have an active story progression system.
Procedure	[A01] Start the game and trigger a story event..
	[A02] Observe dialogue sequences and interactions.
	[A03] Make a choice that affects the story.
	[V01] The game should progress according to the player's choice.
	[V02] Ensure voice acting, subtitles, and animations are synchronized.
Cleanup	Reset the story state if needed

7.3 FUN.01 - Fun Gameplay

TC_ID	FUN.01
Purpose	Measure player engagement and enjoyment.
Requirements	6.3
Priority	Medium
Estimated Time Needed	15 Minutes
Dependency	Core gameplay mechanics should be functional.
Setup	A complete gameplay loop should be available.
Procedure	[A01] Play the game for 15 minutes.
	[A02] Evaluate pacing, difficulty, and challenge balance.
	[A03] Check if rewards, interactions, and progression feel satisfying.
	[V01] Players should not feel frustrated due to poor mechanics.
	[V02] The game should encourage continued play (addiction factor).
Cleanup	Close the game and analyze feedback.

7.4 UI.01 - Easy to Adapt User Interface

TC_ID	UI.01
Purpose	Ensure that UI elements are easy to navigate and understand.
Requirements	6.4
Priority	High
Estimated Time Needed	6 Minutes
Dependency	The UI should be fully implemented.
Setup	The game menu, HUD, and interaction panels should be accessible.
Procedure	[A01] Navigate through the main menu and settings.
	[A02] Play a mission and observe UI responsiveness.
	[A03] Change settings (graphics, controls, sound).
	[V01] The UI should be intuitive and responsive.
	[V02] All text, icons, and buttons should be clearly visible and accessible.
Cleanup	Reset UI settings to default.

7.5 MAP.01 - Optimized Map Creation Phase

TC_ID	MAP.01

Purpose	Ensure that the game generates maps efficiently without performance issues.
Requirements	6.5
Priority	High
Estimated Time Needed	12 Minutes
Dependency	Procedural generation should be optimized.
Setup	The game should be run on various hardware configurations.
Procedure	[A01] Generate a new map and measure load time.
	[A02] Move around the map and observe performance.
	[A03] Check for lag spikes, rendering issues, or missing assets.
	[V01] Map generation should not exceed a certain time threshold.
	[V02] FPS should remain stable without significant drops.
Cleanup	Close the game and analyze performance logs.

7.6 MG.01 - Various Mini-Games

TC_ID	MG.01
Purpose	Verify that mini-games function correctly and provide rewards.
Requirements	6.6
Priority	Medium
Estimated Time Needed	8 Minutes
Dependency	Mini-games should be fully implemented.
Setup	Access to different mini-games in the game.
Procedure	[A01] Start a mini-game.
	[A02] Play through the game and attempt to win.
	[A03] Check if the game provides the expected reward.
	[V01] The mini-game should function without errors.
	[V02] Rewards should be correctly granted to the player.
Cleanup	Reset mini-game progress.

7.7 AI.01 - Smart AI Behaviors

TC_ID	AI.01
Purpose	Ensure AI can react dynamically to the environment and player actions.
Requirements	6.7
Priority	High
Estimated Time Needed	10 Minutes
Dependency	AI system should be fully functional.
Setup	The game should have active AI characters.
Procedure	[A01] Spawn an AI unit in an open area.
	[A02] Move towards the AI and observe its reaction.
	[A03] Attack the AI and observe its combat response.
	[V01] AI should react dynamically to the player's actions.
	[V02] AI should be able to seek cover, attack, or flee intelligently.

Cleanup	Reset AI behaviors if needed.

7.8 FIGHT.01 - Player and AI Combat

TC_ID	FIGHT.01
Purpose	Ensure combat mechanics are balanced and smooth.
Requirements	6.8
Priority	High
Estimated Time Needed	12 Minutes
Dependency	Player and AI combat systems should be fully implemented.
Setup	The game should allow combat between players and AI.
Procedure	[A01] Engage in combat with an AI opponent.
	[A02] Use different weapons and abilities.
	[A03] Observe AI combat patterns and strategies.
	[V01] AI should provide a fair challenge and respond appropriately.
	[V02] Player attacks and dodging mechanics should be smooth and responsive.
Cleanup	Restart the combat scenario if necessary.

TEST RESULT

1. Test Objectives

The purpose of testing the Spaceborn project is to ensure that all game functionalities—including gameplay mechanics, AI behaviors, UI elements, save/load operations, and system performance—meet the requirements defined in the SRS and operate reliably across target platforms.

2. Test Environment

- **Game Engine:** Unreal Engine 5.4
- **Programming Languages:** C++, Blueprint, Python (for ML)
- **Test Systems:**
    - **System A:** Windows 10, Intel i5-9600K, 16GB RAM, NVIDIA GTX 1660
- **Testing Tools:** Unreal Engine Debugger, In-game Console.

3. Tested Features

Feature	Test Case Description	Expected Result	Result
Save/Load System	Save current progress (inventory, AI state, completed objectives) and load it after restarting the game	Game resumes from exact save state	Passed
Procedural Map Generation	Start a new game multiple times to test whether unique layouts are generated every session	Distinct spaceship map generated each time	Passed
Resource Mini-Games	Interact with oxygen, food, energy mini-games and complete them	Game correctly adjusts resource levels	Passed
Machine Learning Enemy Behavior	Allow AI enemy to encounter player repeatedly and observe adaptive behavior	Enemy becomes more efficient in tracking player movements	Passed
In-Game Menu Navigation	Navigate menus: New Game, Load Game, Settings, Exit	Buttons function correctly and redirect appropriately	Passed
UI/UX Responsiveness	Click on in-game UI buttons for construction/defense/resource management	Relevant menu opens; action is executed	Passed
Colonist Status System	Monitor colonist morale, health, and productivity	Colonist data is updated dynamically	Passed
Combat Mechanics	Deploy turrets or traps, engage mutant enemies	Mutants are damaged and react accordingly	Passed
Error Handling - Corrupted Save	Attempt to load a manually corrupted save file	System shows error and returns to main menu	Passed
Performance Under Load	Spawn 5 AI enemies simultaneously and trigger events	Game runs at >30 FPS without input lag	Passed

Feature	Test Case Description	Expected Result	Result
Autosave System	Complete key mini-games and check whether the game autosaves	Autosave file is generated; resumes from correct point	Passed
Settings Menu Functionality	Change resolution, volume, display mode, then save settings	Settings are retained and applied correctly	Passed
Accessibility Features	Enable colorblind mode and test high contrast UI	UI adapts accordingly	Passed
Point Cloud View Functionality	Trigger exploration to display point cloud data	Data visualized in 3D; Zoom and Pan features operational	Passed

4. Bug Reports & Fixes

Bug ID	Description	Status	Resolution
B-101	Save slot 3 failed to overwrite old file	Fixed	Added overwrite confirmation dialog
B-102	AI enemy getting stuck near closed doors	Fixed	Improved navigation mesh generation
B-103	Volume settings not persisting between sessions	Fixed	Linked UI changes to persistent config file
B-104	Mini-map not updating after certain events	Fixed	Added real-time map state refresh
B-105	Embryo status UI overlapping on low resolutions	Fixed	Adjusted HUD scaling and anchors

5. Conclusion

All core game systems have passed functional, usability, and performance testing. Identified bugs were resolved, and the game meets all defined requirements in the SRS. Additional improvements were implemented during development to enhance the overall quality and stability of *Spaceborn*.

USER MANUEL

1. Introduction

Welcome to Spaceborn, a single-player, survival-based space exploration game set in a hostile galaxy. In *Spaceborn*, players must navigate through procedurally generated environments, manage critical life-support resources, and survive dynamic threats from AI-controlled alien forces. This manual guides players through the setup, controls, gameplay mechanics, and troubleshooting.

---

2. System Requirements

Minimum:

- OS: Windows 10 (64-bit)
- Processor: Intel i5-8400 / AMD Ryzen 5 2600
- Memory: 8 GB RAM
- Graphics: NVIDIA GTX 1050 Ti / AMD RX 560
- Storage: 5 GB available space

Recommended:

- Processor: Intel i7-9700K / AMD Ryzen 7 3700X
- Memory: 16 GB RAM
- Graphics: NVIDIA RTX 2060 / AMD RX 5700
- SSD recommended for faster loading

---

3. Installation

Step 1: Download the Game

Visit the official Spaceborn website or game distribution platform (e.g., Steam, itch.io) and download the latest version of the game.

Step 2: Extract Files (if applicable)

If the game is provided as a compressed .zip file, extract it to a folder of your choice**.**

Step 3: Launch the Game

Navigate to the extracted folder and double-click the Spaceborn.exe file to start playing.

Note: You do *not* need to install Unreal Engine, Visual Studio, or any other development tools. Everything you need to run the game is already bundled.

---

4. User Interface

- Main Menu – Start a new game, continue a previous session, access settings or help.
- In-Game HUD – Displays health, oxygen, power, inventory, and environmental alerts.
- Map Interface – Shows your location, mission markers, and nearby threats.
- Terminal Panels – Used to interact with world elements like doors, systems, and logs.

---

5. Basic Functions

- **Start Game**: From the main menu, select “New Game” to begin a new session or “Continue” to load progress.
- **Settings:** Adjust graphics, audio, difficulty level, and key bindings.
- **Save & Load**: Press Esc in-game to save your progress or return to the main menu.

---

6. Running the Simulation

Step 1: Choose Your Scenario

Start a new session. A procedurally generated space environment will be created based on your difficulty and preferences.

Step 2: Survive and Explore

- Gather resources (oxygen, power, food)
- Explore derelict stations and alien-infested zones
- Use terminals and consoles to activate systems, open paths, and avoid threats

Step 3: Manage Your Resources

Keep an eye on your oxygen, energy, and health levels. Use items and equipment found in the environment to survive longer.

---

7. Performance Evaluation

After each session, performance is evaluated based on:

- Survival Time: How long you lasted before being overwhelmed.
- Systems Repaired: Number of critical systems (e.g., oxygen, power) restored.
- Enemy Encounters Survived: Encounters with AI enemies survived or avoided.
- Resource Efficiency: How well you managed life support resources.

---

8. Troubleshooting

Problem	Solution
Game won't start	Ensure that your system meets the minimum requirements.
Low FPS or lag	Lower the graphics quality from the settings menu.
No sound	Check your audio output device and in-game volume settings.
Controls not responding	Ensure no external controller or device is interfering.
Game crashes occasionally	Try updating your GPU drivers and restarting your PC.

---

9. Frequently Asked Questions

Q: Do I need to install Unreal Engine or any other software?
A: No. Spaceborn comes with everything needed to run the game.

Q: Can I play offline?
A: Yes. An internet connection is only required for downloading the game and updates.

Q: Are game saves automatic?
A: The game features both manual and auto-saving systems.

Q: Is multiplayer supported?
A: Not currently. Spaceborn is a single-player experience.